Aalto University
School of Science
Degree Programme in Security and Mobile Computing

Maneesh Chauhan

Measurement and Analysis of Networking Performance in Virtualised Environments

Master's Thesis
Espoo, 30 June, 2014

Supervisors: Professor Antti Ylä-Jääski, Aalto University
Associate Professor Markus Hidell, KTH Royal Institute of Technology
Advisor: Yu Xiao D.Sc. (Tech.), Aalto University

Aalto University
School of Science
Degree Programme in Security and Mobile Computing

ABSTRACT OF
MASTER'S THESIS

Author: Maneesh Chauhan

Title:
Measurement and Analysis of Networking Performance in Virtualised Environments

Date:	30 June, 2014	**Pages:**	74
Major:	Data Communication Software	**Code:**	T-110
Supervisors:	Professor Antti Ylä-Jääski Associate Professor Markus Hidell		
Advisor:	Yu Xiao D.Sc. (Tech)		

Mobile cloud computing, having embraced the ideas like computation offloading, mandates a low latency, high speed network to satisfy the quality of service and usability assurances for mobile applications. Networking performance of clouds based on Xen and Vmware virtualisation solutions has been extensively studied by researchers, although, they have mostly been focused on network throughput and bandwidth metrics. This work focuses on the measurement and analysis of networking performance of VMs in a small, KVM based data centre, emphasising the role of virtualisation overheads in the Host-VM latency and eventually to the overall latency experienced by remote clients. We also present some useful tools such as Driftanalyser, VirtoCalc and Trotter that we developed for carrying out specific measurements and analysis. Our work proves that an increase in a VM's CPU workload has direct implications on the network Round trip times. We also show that Virtualisation Overheads (VO) have significant bearing on the end to end latency and can contribute up to 70% of the round trip time between the Host and VM. Furthermore, we thoroughly study Latency due to Virtualisation Overheads as a networking performance metric and analyse the impact of CPU loads and networking workloads on it. We also analyse the resource sharing patterns and their effects amongst VMs of different sizes on the same Host. Finally, having observed a dependency between network performance of a VM and the Host CPU load, we suggest that in a KVM based cloud installation, workload profiling and optimum processor pinning mechanism can be effectively utilised to regulate network performance of the VMs. The findings from this research work are applicable to optimising latency oriented VM provisioning in the cloud data centres, which would benefit most latency sensitive mobile cloud applications.

Keywords: virtualisation, kvm, cloud computing, network performance
Language: English

Sammanfattning

Mobil cloud computing, har anammat idéerna som beräknings avlastning, att en låg latens, höghastighetsnät för att tillfredsställa tjänsternas kvalitet och användbarhet garantier för mobila applikationer. Nätverks prestanda moln baserade på Xen och VMware virtualiseringslösningar har studerats av forskare, även om de har mestadels fokuserat på nätverksgenomströmning och bandbredd statistik. Arbetet är inriktat på mätning och analys av nätverksprestanda i virtuella maskiner i en liten, KVM baserade datacenter, betonar betydelsen av virtualiserings omkostnader i värd-VM latens och så småningom till den totala fördröjningen upplevs av fjärrklienter. Wealso presentera några användbara verktyg som Driftanalyser, VirtoCalc och Trotter som vi utvecklat för att utföra specifika mätningar och analyser. Vårt arbete visar att en ökning av en VM processor arbetsbelastning har direkta konsekvenser för nätverket Round restider. Vi visar också att Virtualiserings omkostnader (VO) har stor betydelse för början till slut latens och kan bidra med upp till 70 % av rundtrippstid mellan värd och VM. Dessutom är vi noga studera Latency grund Virtualiserings Omkostnader som en nätverksprestanda och undersöka effekterna av CPU-belastning och nätverks arbetsbelastning på den. Vi analyserar också de resursdelningsmönster och deras effekter bland virtuella maskiner i olika storlekar på samma värd. Slutligen, efter att ha observerat ett beroende mellan nätverksprestanda i ett VM och värd CPU belastning, föreslår vi att i en KVM baserad moln installation, arbetsbelastning profilering och optimal processor pinning mekanism kan användas effektivt för att reglera VM nätverksprestanda. Resultaten från denna forskning gäller att optimera latens orienterade VM provisione i molnet datacenter, som skulle dra störst latency känsliga mobila molnapplikationer.

Nyckelord: virtualisering, kvm, cloud computing, nätverksprestanda

Acknowledgements

I wish to express my gratitude to Professor Antti Ylä Jääski and Associate Professor Markus Hidell for their valuable comments and encouraging remarks while supervising this master's thesis. I would like to thank Dr. Yu Xiao for introducing me to the topic and her guidance and encouragement that made this a remarkable learning experience. I would also like to thank the IT staff at the CS&E department for their assistance and support. I am also grateful to my friends and colleagues for facilitating a healthy work environment.
I am thankful to Ms. Aino Lyytikäinen, NordSecMob Program Study Coordinator at Aalto, for her timely reminders and support regarding the thesis practicalities.
Finally, I would like to add a word of gratitude for my parents who have been the inspiration and guiding light all my life and my sisters for their unconditional love and belief in me. Their blessings have given me the strength and hope to successfully accomplish this endeavour.

Thank you

Espoo, 30 June, 2014

Maneesh Chauhan

Abbreviations and Acronyms

AMD-V	AMD Virtualization
API	Application Programming Interface
AWS	Amazon Web Services
BPF	BSD Packet Filter
BSD	Berkeley Software Distribution
EC-2	Elastic Compute Cloud
HD	High Definition
HPC	High-Performance Computing
HTTP	Hyper Text Transfer Protocol
IaaS	Infrastructure as a service
ICMP	Internet Control Message Protocol
Intel VT	Intel Virtualization Technology
IP	Internet Protocol
IT	Information Technology
KVM	Kernel Virtual Machine
LAN	Local Area Network
MTU	Maximum Transmission Unit
PaaS	Platform as a service
QEMU	*Quick Emulator* is a generic and open source machine emulator and virtualizer.
RAM	Random Access Memory
RTT	Round trip time
SaaS	Software as a service
SCP	Secure Copy
SDK	Software Development Kit
SSH	Secure SHell
TCP	Transport Control Protocol
UDP	User Datagram Protocol

VLC	VideoLan Client, is a portable, free and open-source, cross-platform media player and streaming media server written by the VideoLAN project
VM(s)	Virtual Machine(s)
VMM	Virtual Machine Monitor
VO	Virtualisation Overheads
WLAN	Wireless Local Area Network
WiFi	Wireless local area network (WLAN) products that are based on the Institute of Electrical and Electronics Engineers' (IEEE) 802.11 standards

Contents

List of Figures

Chapter 1

Introduction

The divide between the mobile devices and their desktop counterparts has been reducing gradually as most of the latest high end smart-phones come equipped with quad-core processors, gigabytes of memory and high speed wireless connections whether it be over WiFi or 4G-LTE. However promising the trend may seem, the mobile devices are yet to overcome their intrinsic limitations of computational power, network bandwidth and energy constraints. Recent research efforts have led to convergence of mobile computing applications with cloud computing paradigm and it has been touted as a potential solution to overcome the above limitations.

Cloud computing refers to the computational model that offers on demand provisioning of services, platform and computational infrastructure to clients over the Internet. Most of these computational resources, storage and networking infrastructure are hosted in high tech data centres by employing virtualisation on high end servers. In this model consumers can request for virtual machines from the cloud provider with a predetermined configuration as per requirement. The consumers get to rent the computational resources from the cloud providers for their own business or personal use without having to invest heavily in a data centre up front. Furthermore, the elastic nature of cloud service makes major business sense for start-ups and even established companies which aim to rapidly test or even deploy their web based services.

Cloud providers such as Amazon and Rackspace provide IaaS service by packaging preset VM configurations for different business use cases such as a micro , medium or large instances. Also on offer are configurations promising enhanced network performance, disk I/O rates and CPU performance. However, in practice these cloud services are hosted in large data centres

in geographical locations that may even be at cross continental distances. Thus, the network performance may be limited by the sheer distance or hops from the clients to the servers on the cloud. Furthermore, the service level agreements guarantee the allocation of virtual machines with the agreed upon configurations but are silent on the quality of service in terms of guaranteed latency and bandwidth metrics. Furthermore, cloud providers may employ either of the widely popular virtualisation solution like Citrix Xenserver, Red Hat Enterprise Virtualisation , Vmware VSphere and Microsoft Server Hyper-V, each with its own set of advantages and drawbacks. However, virtualisation of the underlying hardware resources also results in performance trade-offs as the Virtual Machines essentially share and even compete for resources such as CPU, memory and network. Therefore, in its current form, the cloud computing paradigm may not be suitable for some emerging mobile computing applications which require guaranteed low latency and high bandwidth network connections.

As the mobile clients are oblivious to the background activities at the server, they expect a low latency connection and high performance from the application. Thus, it is imperative from the Cloud provider's point of view that an explanation for any reduced performance be arrived at so that an acceptable quality of service can be assured. Currently, KVM-Qemu has emerged as a high performance, open source solution for provisioning fully virtualised guests in a cloud environment e.g., Crosspeer Cloud [1] . A great amount of research has been done on competing virtualisation technologies like Xen and Vmware with respect to their networking performance and efficiency. However, very less work has been done on KVM networking performance analysis. Therefore, there is a need for experimentation and analysis to understand and benchmark the networking performance of a KVM-Qemu based IaaS Cloud Service in a mobile cloud computing scenario.

The concept of *Cloudlets* [27] has been proposed by researchers at Carnegie Mellon university, according to whom, "A Cloudlet is a new architectural element that arises from the convergence of mobile computing and cloud computing. It represents the middle tier of a 3-tier hierarchy: mobile device — cloudlet — cloud"[2]. These Cloudlets are built using standard cloud technologies and are designed to have low maintenance overheads while being powerful, well connected, safe, and logically proximate to the mobile clients. This may be a catalyst for many exciting use cases for mobile cloud appli-

[1] http://www.crosspeer.com/
[2] http://elijah.cs.cmu.edu/

cations that would benefit from a low latency service enhanced with high performance computing, data caching and secure intermediate storage at the Cloudlet.

It is an intuitive notion that with an increase in tenancy and workload at the Cloud server, the network performance between the Host to VM and eventually Client to VM link should suffer. To validate this claim, there is a need to study the behaviour of the end to end connection and more importantly Host to VM network peculiarities in a KVM-Qemu based Cloudlet like installation. The effect of factors such as Host CPU workload, Network load, VM configurations and allocation strategies on the network performance of the Cloudlet setup needs to be analysed in detail. Furthermore, the effect of these factors and also the contribution of virtualisation overheads to end to end latency for a client to VM connection has to be analysed through empirical analysis.

1.1 Problem statement

Through this work we aim to answer some of the following questions regarding a KVM-Qemu based mini data centre that has certain features of CMU Cloudlets like high bandwidth and computation power, and logical proximity to the client devices.

1. What are the factors that affect the end-to-end networking performance in a mobile cloud computing scenario and how to quantify the joint effects? This problem requires an understanding of factors that have an impact on each performance metric and whether there is interdependency between these factors. Regarding the metrics of networking performance, we focus on throughput and latency. We consider the end-to-end communications between client devices and the VMs running on the Cloudlet. The end-to-end communications include wireless networking between mobile devices and Cloudlet and the communications between host and VMs.

2. How do added overheads of virtualisation impact the end-to-end network latency for a Virtual Machine ? What is the contribution of virtualisation overheads incurred while forwarding the network traffic from host to VM on the end to end latency of the client to VM connection ?

3. How does multi-tenancy and cross flows from neighbour VMs impact the network performance of a Virtual Machine ? Does an increase in the number of hosted VMs impact the networking performance of the VM under observation ?

4. How is network bandwidth distributed amongst different Virtual Machines on a host ? Is the distribution fair or do VMs with a better hardware configuration starve out lesser VM instances.

5. How do different workloads and VM configuration affect the end to end latency and overall networking performance of a VM ?

1.2 Scope

Measurement and analysis of networking performance in virtualised environments is an enormous pool of research problems and corresponding activities to solve them. Substantial research work has already been carried out on Cloud infrastructures with respect to their networking performance with numerous publications chronicling their results. We limit the scope of this work to solely focus on a KVM-Qemu based data centre that share certain features with Cloudlets, such as high network bandwidth and computation power, and logical proximity to the client devices. The client devices include workstations connected to the test Cloudlet via LAN and Mobile devices that have wifi access. Many researchers have already studied the networking performance of connections over LAN as well as wifi thoroughly, therefore, we would focus solely on the Cloudlet Host to VM network peculiarities in our work. This would give us an idea of the effect of virtualisation overheads on networking performance of VMs in a KVM based Cloudlet setup.

1.3 Methodology

Our research work involves gathering information based on observations and results from experiments carried out on an actual test setup. This follows "Empirical Research" methodology which essentially is a way of gaining knowledge by means of direct and indirect observation or experience. Under this methodology, we first begin with a research question, e.g. "Does an increase in CPU utilisation at a VM in a Cloudlet affect the network connection to a remote client". We have certain theories or notions regarding the research question and we formulate a hypotheses based on them, e.g. "With

an increase in CPU utilisation at the VM, it would experience degraded network performance". Once we have a hypotheses, we then make predictions regarding certain specific scenarios, e.g. "The services running on the VM should experience an increased network latency and even reduced network throughput". Then we design appropriate experiments and configure the test setup to run those experiments with an aim to generate quantitative results that would either support or negate the theories on which our hypotheses are based.

This methodology can be defined as being a cycle of five steps, namely, Observation, Induction, Deduction, Testing and Evaluation. While observation pertains to gathering information and empirical facts to help formulate a hypothesis, we draw up a hypothesis in the induction step. Then we deduce the consequences of the said hypothesis into concrete predictions that can be tested via experiments. Then we carry out empirical investigation through testing and experimentation followed by an evaluation of the experimental results to conclude whether the hypothesis is correct or not.

1.4 Structure of the Thesis

The thesis report is sub-divided into six parts including this introductory section. The next section discusses relevant background information regarding virtualisation, Kernel Virtual Machine (KVM), its networking modes and management mechanism. It explores the emergence of latency as a new performance bottleneck while discussing the use of packet tracing as a means for network performance measurement. It also presents a literature review of related research work.

The section Experiment Design and Implementation describes the design criteria of the test cases and their configuration details. It also discusses the metrics that we measured along with the tools that we developed for measurement and analysis.

It is followed by the section Results and Analysis which elucidates the key findings of our experiments and empirical analysis. It is followed by a discussion of these findings with respect to real world use cases and applications. It also discusses future work and the environmental and social impact of this thesis with respect to green computing.

The concluding section summarises and briefly presents the key findings of

the thesis.

Chapter 2

Background

This section presents some relevant background information and discusses related work which inspired this thesis work. This would help the reader the understand the key theme and concepts of the thesis. Use cases for mobile cloud applications have been discussed followed by a discussion of KVM and its networking modes. As majority of this work is focused on determining the effect of virtualisation overheads on latency, it becomes important to discuss what are the reasons that make latency such a crucial research question. Therefore, this section also discusses the emergence of latency as the new performance bottleneck. It also discusses briefly network performance analysis through active and passive analysis of the network.

2.1 Mobile Cloud Applications

Mobile applications can outsource the computationally and memory intensive tasks to cloud services that offer rich benefits of virtually unlimited computational resources. Such applications can be called *mobile cloud applications*. Cloud based personal digital assistant services have become widely popular with offering from all the major mobile operating systems developers. Notable examples are Siri by Apple, Google Now and the recently unveiled Microsoft Cortana. Recently concluded Apple's Worldwide Developer Conference (WWDC) presented their *icloud* based ecosystem that aims to put the mobile devices at the centre of user's digital experience. Innovative features such as SmartKit for home automation, Healthkit and touchID authentication system may in fact aim at personalising and enhancing our interactions with the world we live in.

Cloud based applications have gathered momentum in spheres of augmented

reality and virtual reality, aided in part by offerings like Qualcomm's Vuforia SDK [1] for cloud based AR apps. Furthermore, recent takeover of Oculus by Facebook inc has fuelled expectations of virtual reality based social networking experience from mobile devices.Furthermore, even the gaming world is warming up to the concept of cloud gaming with offerings like Nvidia Grid [2] and Onlive Cloud Gaming Services [3] that allow users to play latest video games on their mobile devices.

Along with the current commercial applications, there are +several research themes tied to the mobile cloud applications and one of them is the futuristic notion of Cognitive Phones [8] where smart phones would be capable of understanding our life patterns and assess our health and well being while providing intelligent assistance.

Another major research push has been in the field of cloud based computer vision, augmented reality or crowd-sourced reality applications[7]. Such applications aim to build upon the central idea of making the cloud process visual data from the smart phones effectively for creative applications.

There has been a considerable amount of research work related to frameworks that enable effective code offload from the mobile device to the cloud such as *maui* [10] and *ThinkAir* [16]. The deployment models for code and computation offloading include deploying a customized VM to the cloud, which would then act as a proxy for the phone and carry out the computationally intensive tasks at the cloud before returning the final results to the user's device. However, the overall performance gains in code offload is dependent on factors like network speed, energy constraints and amount of data to be transferred. Bahl et.al. [6] discuss different innovative services, programming models and interaction models for a mobile cloud computing infrastructure. Through innovative programming models like code offloading, seamless remote execution and low latency middle tier, mobile applications and services can be enabled to utilise the massive capabilities of the cloud.

A complementary idea to code offloading is to reduce latency by provisioning clouds closer to the clients [27]. Often the public cloud providers provision virtual machines which may be several hop counts away from the clients. As latency is directly proportional to the physical distance or hops between two

[1] https://developer.vuforia.com/
[2] http://www.nvidia.com/object/cloud-gaming.html
[3] https://games.onlive.com/

endpoints, such a network topology cannot allow for a low latency connection. Therefore the idea is to provision small scale clouds or Cloudlets, more or less one hop count away from the clients. Such a configuration can achieve high performance gains for code offload based applications and even for other applications which may benefit from response and caching at the Cloudlet. Face recognition[22] and pattern recognition applications have a natural scope in mobile cloud computing due to the rather complex nature of these algorithms and computations and the resources needed to run those.

All these use cases are remarkably sensitive to connection latency. Furthermore, user perceived delays are extremely important in determining the quality of service or efficiency of the above solutions as they involve extensive user interactions.

2.1.1 Commercial Deployment Environments

Commercial cloud providers like Amazon cloud services, Rackspace, Crosspeer cloud etc. provide certain assurances regarding their services via service level agreements or SLA. They assure the customers of up to 99.999% of service up-time and even compensate the customers with certain credit values for the service time lost due to unannounced down-times[4].

Although, the service level agreements for the procured virtual machine instances guarantees a particular hardware configuration vis-à-vis processor speeds, memory and disk space, it does not explicitly provide network performance guarantees. Although they may have certain features like instances with a higher speed network connectivity, however, they do not clearly mention the details regarding maximum achievable bandwidth, round trip times etc. Clearly, providing such guarantees is a task that is near to impossible, given the Cloud model which is built on top of extensive use of hardware virtualisation and resource pooling.

Thus it is important to understand the behavioural patterns of virtual machines with respect to their configurations and resource requirements in a cloud setup. Next we present some key features of commercially available, virtual machine instances of AWS EC2 service that have inspired some of our experiments.

Amazon EC2 instances provide a number of additional features to facilitate

[4]http://www.rackspace.com/about/datacenters/

deployment, management, and scaling the applications. According to the service description provided on their web pages[5], the virtual machine instances that the users can procure from AWS EC2 service can have the following features:

- **Burstable Performance Instances** Amazon EC2 allows users to choose between Fixed Performance Instances and Burstable Performance Instances. Burstable Performance Instances provide a baseline level of CPU performance with the ability to burst above the baseline. Each Burstable instance receives CPU Credits continuously, the rate of which depends on the instance size. These instances accrue CPU Credits when they are idle, and use CPU credits when they are active. One CPU Credit provides the performance of a full CPU core for one minute.

  Many applications such as web servers, developer environments and small databases don't need consistently high levels of CPU, but benefit significantly from having full access to very fast CPUs when they need them. T2 instances of AWS EC2 service are engineered specifically for these use cases. However, for workloads that require consistently high CPU performance for applications such as video encoding, high volume websites or HPC applications, it is recommended that Fixed Performance Instances be used instead.

- **Multiple Storage Options** Amazon EC2 allows you to choose between multiple storage options based on your requirements. Amazon EBS volumes persist independently from the running life of an Amazon EC2 instance. Once a volume is attached to an instance you can use it like any other physical hard drive. It also offers the choice of storage disks from magnetic storage devices to solid state disks that provide enhanced performance. Many Amazon EC2 instances can also include storage from disks that are physically attached to the host computer. This disk storage is referred to as instance store. Instance store provides temporary block-level storage for Amazon EC2 instances. The data on an instance store volume persists only during the life of the associated Amazon EC2 instance. In addition to block level storage via Amazon EBS or instance store, Amazon S3 can also be used for highly durable, highly available object storage.

[5]http://aws.amazon.com/ec2/instance-types/

- **Enhanced Networking** Enhanced Networking enables the users to get significantly higher packet per second (PPS) performance, lower network jitter and lower latencies. This feature uses a new network virtualization stack, that provides higher I/O performance and lower CPU utilisation compared to traditional implementations. In order to take advantage of Enhanced Networking appropriate driver must be installed. Enhanced Networking is currently supported in C3, R3, and I2 instances.

- **Cluster Networking** Instances launched into a common cluster placement group are placed into a logical cluster that provides high-bandwidth, low-latency networking between all instances in the cluster. Cluster networking is ideal for high performance analytics systems and many science and engineering applications, especially those using the MPI library standard for parallel programming. C3, I2, CR1, G2, and HS1 instances support cluster networking.

- **Dedicated Instances** Dedicated Instances are Amazon EC2 instances that run on single-tenant hardware dedicated to a single customer. They are ideal for workloads where corporate policies or industry regulations require that user's EC2 instances be physically isolated at the host hardware level from instances that belong to other customers. Dedicated Instances let the users take full advantage of the benefits of the AWS cloud – on-demand elastic provisioning, pay only for what is used, all while ensuring that the Amazon EC2 compute instances are isolated at the hardware level.

2.2 Latency: A Performance Bottleneck

Most of the Internet service providers offer high speed broadband Internet plans that boldly market the massive amounts of network bandwidth on offer. Having great amounts of bandwidth is definitely advantageous, especially while streaming an HD quality movie or downloading insanely large data files. However, for our general web browsing experience it is the connection latency that appears to be the compromising factor.

2.2.1 End to End Latency

End to end latency or delay refers to the total time taken for a packet to be transmitted over a network from a source to the destination. The overall

time is a summation of sub delays incurred while transmission, propagation and packet processing, among other things, at different links on the path between the source and destination.

$$EndToEndLatency = X[trans\_delay + prop\_delay + proc\_delay + q\_delay]$$

Where trans_delay is the transmission delay, prop_delay is the propagation delay, proc_delay is the processing delay, q_delay stands for queueing delay and X stands for the number of links (routers + 1) on the path between source and destination.

2.2.2 Network Bandwidth vs Latency

Network bandwidth can be considered to be a multiple lane highway between two stations (end points) carrying traffic back and forth. In order to be able to transfer more vehicles, we increase the number of lanes, similarly, in order to achieve higher data rates, we covet a higher network bandwidth. However, the distance between the two end points can rarely be reduced, therefore, the time taken by network traffic to propagate between the two end points is a function of the actual physical distance, and the transport medium.

Akamai's report on the state of the Internet [3] reveals that on average, most of the mobile data users worldwide have access to networks with 4 Mbps speeds. Which means that, most of the users worldwide can already enjoy mobile services requiring considerable network bandwidths without much troubles. Furthermore, Ilya Grigorik [11] states that an increase in bandwidth from 5 Mbps to 10 Mbps results in a mere 5% improvement in webpage load times. Therefore, investing in a high bandwidth plan for these users won't make much different in terms of network performance.

2.2.3 Latency in Clouds

Furthermore, in a Cloud environment, the paradigm of acceptable level of latencies is much different to those perceived in enterprise applications. The perceived latencies may be dependent on the carriers as well as the Cloud infrastructure. Minnear [21] provides an interesting analysis of the latency issue in a Cloud and proposes measures like optimal network monitoring, efficient routing mechanisms and need for robust SLA and so on. There is an apparent shift towards user perceived delays as a prime performance metric for most of the Mobile Cloud applications, with users unwilling to accept a

service that does not meet their standards of user perceived delays.

2.2.4 Towards reducing Latency

Latency is directly proportional to the physical distance and the carrying medium. Therefore, the only means to reduce the connection latency is generally to make shorter wires made of high speed optical fibres. Essentially this means to bring the servers closer to the clients both logically and physically. This is why there is such a wide network of content delivery servers that cache content and respond to queries directly from the edge of the network. Future Internet infrastructures research like Information Centric Networks is adopting innovative ways to reduce latency by distributed caching. Furthermore, ideas like Cloudlets aim to reduce latency by bringing the computation nearer to the clients where, the clients no longer have to incur long propagation delays to the cloud, as a minimalistic setup is available just a network hop away.
Contemporary research in mobile cloud computing involve research domains like speech recognition, natural language processing, computer vision and graphics, machine learning, and augmented reality which employ capturing sensory data from the smart phones in varying degrees. Such application may involve lightweight sensing data from GPS and accelerometer to data heavy pictures, Audio and Video captures. Naturally such applications require a high bandwidth network connection. Furthermore, most of these applications involve active user interactions which renders them extremely sensitive to user perceived delays or latency.
According to John Carmack, one of the foremost game programmers in the world and CTO of Oculus VR, 20 millisecond of latency is the threshold for most of the human in the middle applications such as virtual reality and interactive 3D systems. "Human sensory systems can detect very small amounts of relative delays in visual or audio fields, however when these delays are smaller than 20 ms, they are almost imperceptible" [2]. He also proposes innovations like View bypass and time warping to achieve high performance especially in the field of Cloud Gaming. However these are application layer measures that automatically assume an acceptable performance on the part of the cloud networking infrastructure and aim to build on top of that.
Therefore, finding out factors that may directly or indirectly affect network latency at the VM level or the VM host in the Cloud is an important research question which we aim to answer though this thesis.

2.3 Kernel Based Virtual Machine

Kernel Based Virtual Machine or KVM [15] is a high performance full virtualization solution for Linux based systems. It is available specifically for x86 hardware that support Virtual machine extensions (Intel VT or AMD-V). It comes as a Kernel loadable module, kvm.ko that can be loaded with the *modprobe* command. The guest operating systems are run as separate user mode processes and they can be monitored just like any other process running on the host, using commands such as top or ps. The scheduling tasks are delegated to the linux kernel and Qemu is used for emulation of underlying hardware. Therefore KVM is lightweight, fast and incurs less overheads than its bulkier counterparts like Vmware ESX, Virtualbox, Xen etc.
KVM has full support from the *Libvirt Virtualisation API* which is a toolkit to interact with the virtualization capabilities of recent versions of Linux (and other OSes)[6]. As a result, utilities like *vmbuilder and virsh* can be used to create virtual machines or to interact with the Libvirt API. A graphical user interface based tool called Virt-Manager is also quite useful as it allows for easier provisioning and maintenance of virtual machines under KVM. Earlier KVM and Qemu used to be separate projects with KVM being a fork of Qemu, however recently the two branches have been merged and now Qemu or Qemu-KVM represents the official KVM repository. With its considerable developer base, KVM promises to be the front runner in open source hypervisor solutions. Furthermore, due to its design it is closely linked to the Linux kernel itself, which enables it to provide high performance, security and reliability. Moreover, KVM or recent QEMU (recent than 0.9.1) versions support Virtio [7] virtualisation standard which empowers the guests with greater networking and I/O performance.

2.3.1 KVM Networking Modes

There are a few different configurations or modes to allow virtual machines access to the external networks[8]. The default configuration is the User-mode Networking that uses NAT to transmit traffic through the host network interface to the external network. Bridged mode is another configuration which allows external hosts to access the services running on the guest VMs. Finally, Private bridged mode allows the guest VMs to communicate with each other through a private virtual bridge that is not accessible from the

[6] libvirt.org
[7] http://wiki.libvirt.org/page/Virtio
[8] https://help.ubuntu.com/community/KVM/Networking

external network.

- **User-mode Networking**

  This is the default networking mode under KVM, and it allows a guest, access to network services, Internet or to any resources available over the local network while preventing the guest's visibility to other hosts on the network. In essence, the guest will be able to access the Internet but will not be able to host a publicly accessible web server. User-mode networking does not support a number of networking features like ICMP, therefore, certain applications may not function properly. This is the default networking mode under KVM, and if we just run our guest without specifying any network parameters, a user-level or slirp networking will be created by default. The IP address are assigned automatically to the guest by the DHCP service integrated in QEMU. We don't need to specify a different MAC address for each guest, in case we run multiple guests on the host. KVM-QEMU automatically assigns unique MAC addresses to the guests. Finally, this mode also allows port forwarding between the guest and the host by using the "hostfwd" option. For example, if we specify $hostfwd = tcp :: 5555- : 22$ when we run the guest, the host port 5555 will be forwarded to the guest port 22, which will allow SSH and SCP between the host and the guest.

- **Bridged Networking** Public Bridged networking allows the guests to be accessed from other guests and also the network. The use case for this networking configuration is to allow our guest to have an IP address of the local network to which it has been connected. Furthermore, this networking mode provides maximum performance for our virtual machines.

  This is achieved by connecting to the outside world through the physical interface of the host machine, thus making them appear as actual physical hosts. A virtual bridge is created that bridges the guest networking device to the host networking device. This configuration works well with the Ethernet network cards, however, it does not work if the physical interface being used for bridging is a wireless device.

  Creating a network bridge on the host is a trivial task and there is comprehensive documentation available on-line to this effect [9] [10]. We

[9] http://www.dedoimedo.com/computers/kvm-bridged.html
[10] https://help.ubuntu.com/community/KVM/Networking

need to install the bridge-utils package, following which we need to add the entry for the bridge interface in the */etc/network/interfaces* file. Once the configuration has been done, a restart of the networking service is necessary for the changes to take effect. As a result, the bridge interface will start sharing the host's network interface and will receive an IP address from the network DHCP server. As the bridge is connected to the guest machine, the other hosts on the network can view and communicate with the bridged guests just like any normal host.

- **Private Bridged Networking** This networking mode provides the flexibility of configuring an exclusive private network for two or more guests. These guests will neither be accessible from other guests on the same Host nor from the real network. The MAC addresses for the guests can be generated automatically or they can even be user specified. In any case, the MAC addresses of the guests on such a network must be unique. Such a network can be easily created from the command line using the *Virsh*[11] command line tool.

2.3.2 KVM Scheduling

KVM employs completely fair scheduling (CFS)[1] which was merged into the 2.6.23 release of the Linux kernel and is the default scheduler. The advantage of CFS is that it introduces the concept of sleeper fairness wherein a task that spends much time sleeping is allowed to access comparable CPU time to a continuously running task. This is implemented via a red black tree that holds the planned task records with the spent processor time as the key. This allows the processes that have used least amount of CPU time to be picked efficiently. As soon as the process is picked, its spent time value is updated and it is re inserted in the tree at an appropriate position. As this mechanism takes into account the spent time of tasks, it ensures that a waiting or sleeping task would automatically receive a priority boost when it needs it.

Researchers have been working on fine tuning the scheduling mechanisms in KVM. They have proposed that instead of scheduling vCPUs, processes should be scheduled in order to reduce the burden from the Hypervisor [28] [31]. It remains to be seen when these propositions become part of mainstream KVM.

[11] https://help.ubuntu.com/community/KVM/Virsh

2.4 Network Performance Measurement

Network performance measurement is a necessary activity that requires effort and forethought. As network is a long term investment, there is a need to optimise the capital expenditure as well as the operational expenditure. Several demands like accounting, performance, contractual obligations, regulations and market competition, make it necessary to continuously monitor and maintain the networks. We may need to measure and quantify, network performance and utilisation, user behaviour, traffic demands and even networking equipment performance.

From a cloud provider's point of view, this becomes absolutely necessary in order to verify the SLA requirements, to identify network bottlenecks and optimise the network by moving servers etc., and also to monitor network usage and identify malicious activity or illegitimate network traffic.

Once we have obtained the data sets, we carry out algorithmic data analysis using specialised tools for processing the large data sets and we can construct models or extrapolate data to compensate for certain missing information.

The two common approaches for network performance measurement are the passive and active approaches. These two are complementary to each other and should be used in conjunction, for a comprehensive measurement.

2.4.1 Passive Measurements

Passive measurement means to measure a network, without creating or modifying any traffic on the network [12]. This can be done by introducing specialised hardware such as Sniffer, or can be done by introducing the measurement software at the routers or even the end hosts.

This method reveals details about the network protocol, actual contents of the packets and inter arrival times. Passive measurement provide a means of debugging a network application, by providing a user with the entire packet contents, as seen on the network. This method does not introduce any new traffic in the network, therefore, it does not affect the running applications or the networking infrastructure.

Since the passive approach may require viewing all packets on the network, there can be privacy or security issues about how to access/protect the data

[12] http://wand.net.nz/pubs/19/html/node9.html

gathered. Tcpdump and Wireshark are two widely used tools for passive measurement and analysis of network traffic.

2.4.2 Active Measurement

The active approach employs injection of test packets into the network or sending packets to servers and applications and measuring service obtained from the network. It creates extra traffic, however, the volume and properties of the introduced traffic is fully adjustable and meaningful measurements may be obtained by introducing a small volume of test traffic into the network.

Active measurements provide more flexibility and control on the nature of traffic generation, the sampling techniques, the timing, frequency, scheduling, packet sizes and types (to emulate various applications), statistical quality, the path and function chosen to be monitored [13].

Active means freedom to test what we want and in whichever way we desire. It also simplifies the emulation of real world scenarios and verification of Quality of Service guarantees and SLA agreements is pretty straightforward. Active measurement provides the end to end network state between two hosts with indications of a networks performance as Packet round trip time (RTT), Average packet loss and Connection bandwidth amongst others.

2.4.3 Measuring Latency

ICMP echo request and response messaging utilities like Ping [14] might not be suitable for accurate measurements of network RTT [23]. However, for a network with simpler configuration where machines are essentially one hop away, such utilities may be used. Furthermore, there are more accurate techniques for passively measuring network RTT like the use of TCP timestamp ts_val and ts_ecr values [30] [14]. This technique has been used in the TCP Round Trip Time Calculation tool that we have developed for passively monitoring TCP connections at the Cloudlet.

[13] http://www.slac.stanford.edu/comp/net/wan-mon/passive-vs-active.html
[14] http://manpages.ubuntu.com/manpages/lucid/man8/ping.8.html

2.5 Related Work

In their detailed analysis of various scenarios synergistic with ubiquitous cloud computing, Van Der Merwe et.al. [33] present innovative mechanisms like *towards the sun* and *cloudbursting* in order to achieve high Cloud performance and low network delays. They suggest that basic building blocks exist for realising sophisticated private/public cloud interactions. However a holistic approach is necessary to orchestrate these building blocks, that takes care of cloud conditions and network conditions while honouring user demands.

Virtualisation is at the heart of cloud computing, therefore, it is natural that many research efforts have gone into quantifying the performance of virtualisation infrastructure and virtual machine monitors. Virtual machine monitors or hypervisors like Xen and Vmware have been thoroughly studied in literature with different studies aiming to characterise their networking performance and ways to reduce the impact of virtualisation overhead on network performance of VMs [4] [5] [9].
Aravind Menon and others developed *Xenoprof* [20] , a system-wide statistical profiling toolkit for the Xen virtual machine environment. The toolkit enables coordinated profiling of multiple VMs in a system to obtain the distribution of hardware events such as clock cycles and cache and TLB misses. They propose to use this toolkit to analyse performance overheads incurred by networking applications running in Xen VMs.

Recently, researchers have proposed innovative mechanisms like *Efficient Interrupt Coalescing* and *Virtual Receive-Side Scaling* [12] to improve the network I/O performance in virtual machines. These methods rely on improving the receive interrupt processing by the VMs in order to enhance overall networking performance gains.

Studies have also been conducted that compare the performance and scalability aspects of Xen and KVM virtualisation solutions. Sorija et. al. [29] carried out a comparative study for Xen and KVM and found out that they gave comparable performance, with KVM just about edging out Xen. They tested the impact of increasing the number of VMs on a single physical host and found out that performance took a slight beating when the number of allocated VMs was more than the actual physical CPU cores. We have also incorporated their experimental theme in the form of iterative expansion of multi tenancy in our experiments and have extended it further by imple-

menting processor pinning based experimental scenarios.

Jon Whiteaker et. al. studied the impact of virtualization on round-trip time (RTT) measurements by conducting controlled experiments on Linux-VServer and Xen based setups[35]. They discovered that virtualisation introduced more delays while sending packets and that network intensive workloads on neighbouring VMs introduced delays as much as 100 ms in the round trip times. Therefore, we have tried to study bidirectional characteristics of networking performance metrics in our work and have also designed experiments to study cross effects of neighbouring VMs.

Taking a step further into the Cloud infrastructure, researchers have focused on performance analysis of various network I/O Workloads in clouds and the data centres that host them. Yeiduo Mei et. al. [19] studied the impact of workloads on neighbouring VMs on a newly allocated VM. They also present an in-depth discussion on the performance impact of co locating applications that compete for either CPU or network I/O resources. They also analysed the impact of different CPU resource scheduling strategies and workload rates on the performance of applications running on different VMs hosted by the same physical machine. Studies have also been carried out specific to commercial Cloud Service providers like Amazon EC2 services where researchers studied the effect of virtualisation on the performance of VMs on the amazon EC2 cloud [34].

Resource sharing is a primary tenet of virtualisation, and network is a resource that is shared on a best effort basis. Therefore, researchers like Popa and Krishnamoorthy [24] have studied the effect of network sharing on cloud computing performance and have proposed guidelines to optimise the tradeoffs between payment proportionality and bandwidth guarantees for VMs. On similar lines we also discuss certain guidelines for optimising latency based task placement in the cloud.

Certain task placement frameworks such as Choreo by Lacurts et. al. [17] facilitate efficient placement of computational tasks on a cloud by profiling and testing the configuration of the network topology and the virtual machines available at the cloud. The application to be run on these virtual machines is divided into different tasks which are then placed on the virtual machines depending on the resource requirement of each task and the performance profile of the virtual machines. They measure metrics like inter VM

throughput using Netperf [15] tool to aid in generating an optimum task placement solution. Furthermore, host to VM throughput measurements help in establishing the effect of cross flows on the bandwidth available to the VM under test. Some of our bandwidth measurement experiments are inspired by their work. They have also implemented an algorithm to detect bottleneck links on the path between different allocated VMs by inter VM throughput measurements. However, in our work as all the VMs are essentially hosted on the same server hardware, they are just one hop away from each other and the nearest bottleneck link would be at the host network card.
There has also been an emphasis on reducing the user perceived delays for different interactive mobile apps. It can be achieved through a negotiation mechanism between the client and the server that enforces strict time deadlines for different tasks, thereby, enabling the server to speed up its computations to meet the deadlines [25]. Their work emphasises on the importance of clock synchronisation between the clients and the server in order to generate accurate timing results. Taking a cue from their work we developed a drift analysing tool that we can use to measure the offset between the client and the Host machine clocks so that further experiments are accurate.

[15]http://manpages.ubuntu.com/manpages/karmic/man1/netperf.1.html

Chapter 3

Experiment Design and Implementation

3.1 Test Setup

The test setup required for our experiments must satisfy certain criteria. It must be able to provision virtual machines that resemble or emulate the hardware configuration offered by popular IaaS cloud service providers such as Amazon Web Services and Crosspeer cloud. While it must be capable of delivering appropriate processing power and memory to support such instances, it must also be easily accessible and logically proximate to the measurement points or devices. This is required as we are testing the networking performance of a KVM based, small data centre that resembles certain features of Cloudlets i.e., virtual machines provisioned on a HPC server that is located ideally one hop away from the client devices (This test setup will be referred to as "Cloudlet" henceforth in this report for the sake of convenience). Therefore, it became necessary that the servers used for the experiments were available on the same LAN as the client workstation and also had access via Wifi for mobile devices. We used a workstation on the same LAN as the Cloudlet server as a measurement point along with a mobile device that had access to the Cloudlet over Wifi.

3.1.1 Hardware setup

We intended to follow the configurations adopted by AWS EC2 general purpose instances in our experiments to mirror real world use case. Zhuang [36] also carried out experiments on a local cloud based on Xen hypervisor. Therefore, we have established a local platform based on KVM virtualisation infrastructure to emulate the characteristics of commercially available VM

instances such as AWS EC2 instances. As we required to experiment and test out some of our hypotheses on a test cloudlet setup, we limited the scope of our experiments to only include virtual machine instances containing one to four VCPUs or more colloquially, micro to small VM instances.

We used two physical servers for our testing purposes. Initially we used a workstation with Intel(R) Core(TM)2 Quad CPU Q6600 CPU having 4 processor cores and speed of 2.4GHz and 8GB RAM mainly for testing with micro instances. We also ran experiments on a rack mounted server with a hardware configuration similar to a full server instance of commercial cloud service providers. It contained a 16 core Intel(R) Xeon(R) CPU E5640 running at 2.67GHz speed along with 64GB RAM and support for Gigabit Ethernet.

3.1.2 Software and Programming Environment

The host machines ran a 64 bit version of Ubuntu 12.04 LTS operating system while the guest machines ran Desktop versions of 32 bit Ubuntu 12.04 LTS operating system.

The programming environment for all the tools that we developed was Python 2.7.6 running on a 64 bit Ubuntu 12.04 LTS machine. We also developed some bash shell scripts to automate the experiments and for some minor configurations. For the development of certain tools we required Python modules such as Scapy and Paramiko which had to be installed additionally.

Virt-Manager and Virsh tools were downloaded and installed on the Host machines. We also downloaded and installed latest versions of Open-ssh server and tools like Netperf version 2.6.0 and Tcpdump release 4.6.1 on the Host as well as the guest machines.

We used VLC player to establish a media streaming server on some guests and used a free and open source video [1] as the video source.

3.2 Metrics Studied

The most important metrics while measuring and analysing network performance are the network bandwidth and round trip time. We wished to study bandwidth allocation patterns on the Cloudlet and to analyse the factors

[1] http://sitasingstheblues.com/

affecting this distribution. We considered bidirectional throughput between the Host and the virtual machines and used Netperf to get these metrics.

Furthermore, we measured the round trip times between the Host and the virtual machines and analysed the factors that affect this metric. Most importantly we thoroughly studied the Latency due to Virtualisation Overheads which we measured by using VirtOCalc tool which will be described in the next section.

3.3 Prominent themes in the Experiments

Impact of Cross flows and the CPU workloads on neighbouring workloads is an important influencing factor and has been thoroughly analysed through experimentation in our work. Furthermore, we have focused on studying the bandwidth distribution patterns under different scenarios at the Cloudlet and have tried to analyse the impact of virtualisation overheads on the overall networking performance of the hosted VMs.

Our experiments take inspiration from certain scenarios mentioned in the paper describing the Choreo framework[17]. As our work focuses on the client to Cloudlet network, these experiments have to be adapted to account for the Client–Host–Virtual Machine path and the conditions affecting the network performance in this chain. Most importantly, as the idea is to study the impact of virtualisation overheads on the networking performance, we focus on the Host to VM network because mobile access networks have already been widely studied while network latency in our case is simply caused by Wireless / LAN communication and the virtualisation overheads.

We plan to study the effect of increasing multi-tenancy on the Host, on the network performance of the Test VM. Furthermore, we wish to carry out CPU intensive computations to load one of the VMs while carrying out measurements on the other VM to analyse the effect of Cross CPU load. Another scenario is running a bandwidth grabbing test like HD video streaming on one VM and analysing the effect on the performance metrics of the Test VM.

Incremental measurements technique was followed in our experiments across all scenarios, i,e iterative measurement with an additional guest launched on the same host in each iteration. This would help us understand the effect of launching several VMs on the same physical host and also how the increase

in the number of VMs affects the overall network performance.

3.4 Tools and Techniques

Various tools and utilities were used for collecting data and measurements in the experiments. Some tools are widely used by professionals and researchers alike for benchmarking networking performance while we developed a couple of tools specifically for measurements pertaining to our special use cases. The following subsections describe the different tools followed by the measurement techniques that were used in the experiments.

3.4.1 Drift Analyser

Accurate time measurements result in accurate experimental results. In order to achieve that, the system clocks of the two measurement end points should be synchronised. Clock synchronisation is the most important preliminary step in applications where decision making is dependent upon accurate measurement of arrival and departure time of packets [25]. Virtualisation poses challenges for guest time keeping as some guests may use Time Stamp Counters(TSC) for timing while others may not. As some CPUs do not have a constant TSC, this may result in the guest's clock running either faster or slower than the host system clock. KVM takes care of these problems by providing paravirtualized clock for the guests and also the mechanism of clock synchronisation using network time protocol. Still, to get a high precision time measurement we need a mechanism to calculate the relative clock offset of the guest machine from the host so that appropriate corrections can be applied to the measured reports. Furthermore, as some long running guests may tend to drift from the host system clock, such a mechanism may help in compensating for the clock drift as well.

Drift Analyser is a tool that caters to this requirement as it implements a simple algorithm based on request response packet timestamps as shown in the Figure 3.1. It has been developed in Python and incorporates local as well as remote packet capturing using the Scapy and Paramiko modules. It initiates packet capturing at both the Host as well as the Test VM. Then it starts sending ICMP request messages from the host which are replied to by the guest, while capturing all these packets at both ends. Eventually, it copies the pcap file from the guest to the Host machine and the comparisons are carried out at the Host. The result is the average Clock drift offset rela-

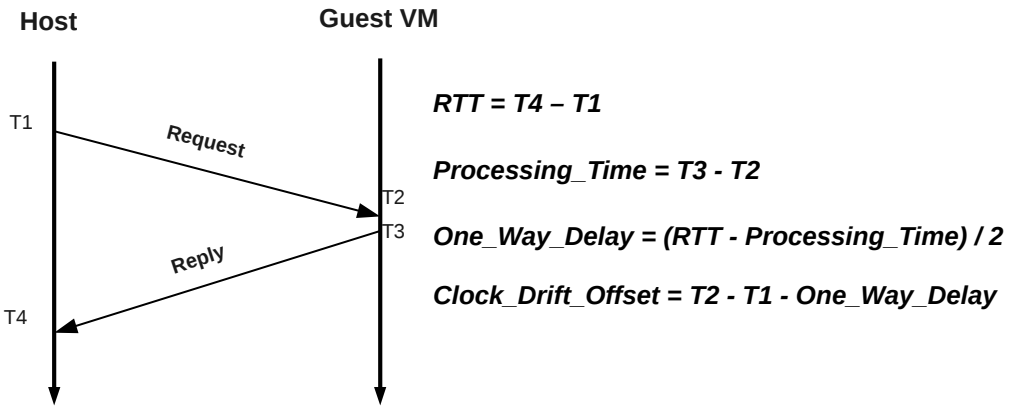

Figure 3.1: A demonstration of the working of the drift calculator tool. It is based on the timestamps of request and response packets which are used in the equations on the right to generate the clock offset value.

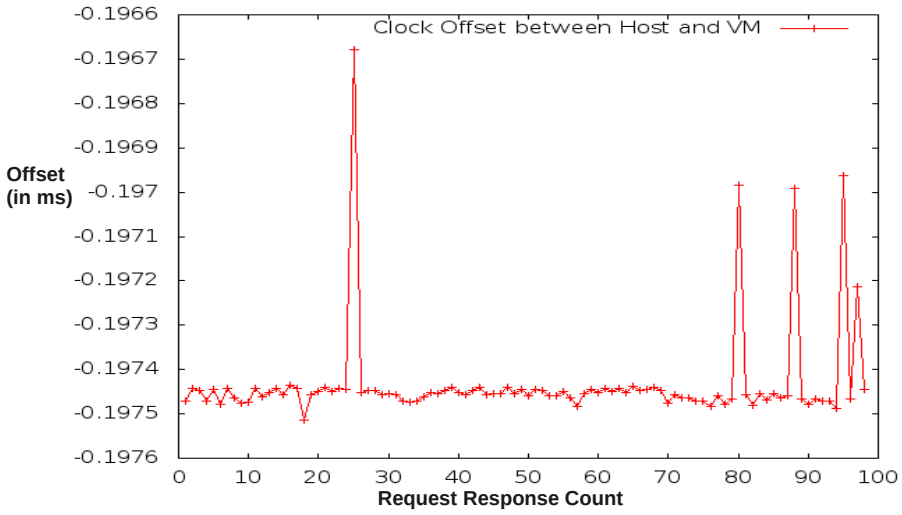

Figure 3.2: A plot of clock offset values generated for each request response pair by the Drift Analyser tool.

tive to the Host machine's system clock. This value has been observed to be fairly stable and is used to adjust the time measurements in the experiments that follow. It has been observed to give a stable output under various workloads and VM configuration on the Host as can be seen in Figure 3.2.

Drift analyser was instrumental in all our experiments as it was run as a preliminary step to ensure accurate measurement of clock offset between the Host machine and the test VM which ensured accuracy of further measurements.

3.4.2 TRoTTEr: TCP Round Trip Time Estimator

Estimating and monitoring round trip times is essential for understanding network performance as well as to measure the performance of services implemented over the network. This information can be utilised by the service providers to fine tune the environment as well as their service to ensure a better quality of service. RTT measurement for TCP flows is important because it carries majority of the network traffic and most of the mobile services that utilise cloud computing are TCP based. Furthermore, the TCP protocol stack can fine tune its performance based on the TCP round trip time estimates. Moreover, this presents the real picture of networking delays as being experienced by the users of the service and measures can be taken to improve the responsiveness of the service if considerable delays are encountered.

Active probing is one approach to get these estimates, however, this approach adds more traffic to the network and such luxuries are ill afforded in a cloud scenario where networking resources are limited due to extensive sharing. Secondly, we could implement some application layer mechanism to reflect the round trip times experienced by the packets over the network. This approach also adds a caveat of being application specific and adds overhead on the performance of the service itself. Therefore, we must make do with whatever existing traffic we may encounter using passive analysis techniques.

Therefore, we developed a tool for estimating TCP round trip times which has been aptly named TRoTTEr. It is based on a paper by Stephan Strowes in which he talks about a mechanism for passively measuring TCP round trip times [30]. This method is based on the RFC 1323 as authored by Jacobson et. al. [14] that presents a method for calculating TCP round trip times using the TCP timestamp options.

Underlying Mechanism

TCP timestamp field is contained in the TCP header of most of the traffic nowadays although it is not mandatory. The timestamp values are held in two 4-byte header fields: ts_val (Timestamp Value) and ts_ecr (Timestamp Echo Reply). Whenever a TCP segment is transmitted from one host to the other the ts_val field is also included and the receiving host has to include corresponding ts_ecr in return. The time difference measured between first emitting a ts_val and receiving it in a ts_ecr provides us an estimate of the RTT. Timestamp is an arbitrary value that is incremented according to the system clock, however, it has no meaning independently.

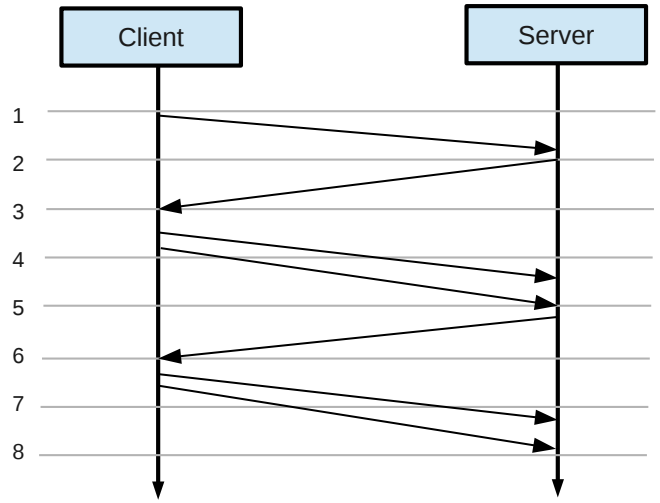

Figure 3.3: A simplified demonstration of TCP timestamp exchange between two hosts namely, client and server

We can understand this method by way of an example. In the Figure 3.3 time progresses from top to bottom, and the horizontal lines indicate real times in milliseconds. There is an open connection between client and the server and they are exchanging packets. We can assume that the clocks at both ends are synchronised. Let us assume that the client first emits a TCP segment that contains the timestamp options:

$$ts\_val = 1, ts\_ecr = 0$$

Here ts_ecr is set to 0 because till now now ts_ecr has been received from the server to the client. This usually indicates that the client is initiating a

connection to the server. The server receives this timestamp at time 1 while at time 2, it emits a TCP segment back to the client containing the values:

$$ts\_val = 2, ts\_ecr = ts\_val\_client = 1$$

These are received at the client at time 3. Given this received value and the current time, client knows that RTT in this instance is approximately 2 ms. Subsequently, the next two segments that the client emits, both carry the values:

$$ts\_val = 3, ts\_ecr = ts\_val\_server = 2$$

The first of these is received at the server at time 4, so the server can also calculate an RTT of 2 ms. In case of multiple packets carrying the same ts_ecr values, the timestamp of the packet received earliest gives the closest estimate of the RTT experience over the network.

A limitation of this method is that both the hosts need to transfer data to each other in order to be able to calculate RTT. In case the flow is only unidirectional, only one host will be able to calculate RTT. We have used this tool in our experiments to calculate the TCP round trip times for TCP flows between a client machine and a Server hosted in a cloudlet over a local area network.

Validation
In order to validate this tool the following experiment was carried out. A TCP connection between the two hosts on the same LAN was set up using netcat utility and then TRoTTEr was run to monitor the connection. Several iterations were carried out, once with 100 measurements and then with 500 RTT measurements. To validate the measurements, 100 and 500 ping requests were made from one host to the other simultaneously when TRoTTEr was running and the resulting observations have been listed in Table 3.1. After several iterations of the experiment it became clear that the RTT measurement tool provides an accurate estimate of the TCP round trip times and the results match those obtained by using a comparable configuration of ping command between the two hosts. This experiment was carried out between two hosts on the same network while no other substantial network traffic flowed between them. Furthermore, as the timestamp fields in the TCP header are present by default and are widely supported by most of the protocol implementations, this method does not incur major overheads. This method is perfect for passive measurements and analysis of TCP traffic and it makes the TRoTTEr tool quite useful and relevant for our experiments.

RTT	P100	T100	P500	T500
Minimum	0.278	0.263	0.259	0.255
Average	0.563	0.551	0.567	0.572
Maximum	0.679	0.629	0.710	1.641
Std. Dev.	0.045	0.069	0.052	0.083

Table 3.1: A comparison of results obtained using TRoTTEr with a parallel run of ping command. P100 and P500 refer to the values obtained by transmitting 100 and 500 ICMP echo request reply pairs respectively. Whereas, T100 and T500 refer to the results obtained from TRoTTEr for 100 and 500 TCP packets respectively.

3.4.3 VirtOCalc: Tool to Calculate network latency due to VO

It is widely known that the development of hardware and software virtualisation revolutionised the Software and IT industry and ushered in the era of cloud computing. Virtualisation enables optimum utilisation of hardware resources and minimises resource wastage. If enough network and storage resources are provisioned, a server with a decent hardware configuration can be used to provision multiple virtual machines that could in theory, provide a better overall performance than the single host. However, virtualisation carries with it the added burden of processing overheads which are a natural result of sharing the host hardware and network resources between the guest machines. In certain cases over-subscription of the underlying resources by virtual machines may result in delays and scheduling conflicts that directly affect the performance of the individual virtual machines. A relevant case is that of a soft real time mobile application that depends on the server on the cloud for mission critical computations. We need to find out how is the end to end network latency between the server and the mobile device being affected by the variation in the VM allocation and overall resource usage at the cloud host.

An important thrust of our research is to find out how virtualisation overheads affect the end to end latency for an application hosted on the cloud. To achieve this goal we have developed the tool **VirtOCalc** that provides precise measurements of the time taken by the packets between the virtual machines and the host.
The end to end network latency between two hosts or a server and a client comprises of different sub components, as was described in Section 2.2. We are most interested in measuring the latency due to virtualisation overheads

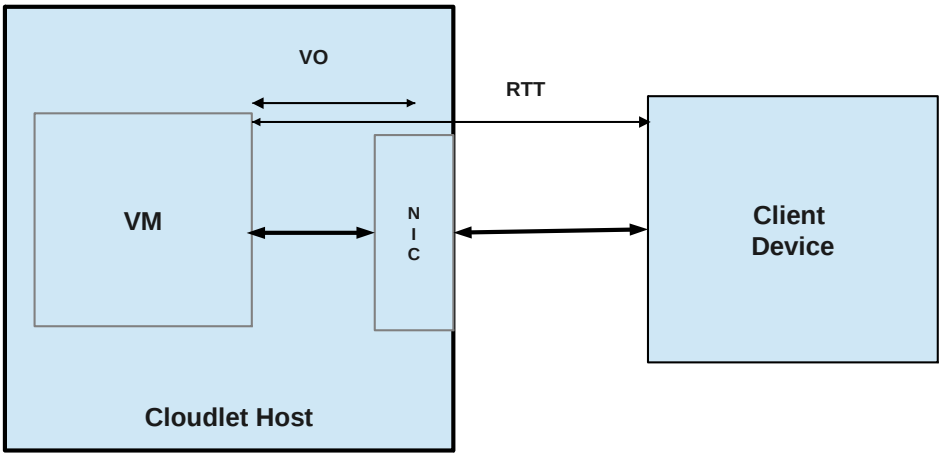

Figure 3.4: General data flow between client and the VM with a representation of what we mean by virtualisation overheads in this flow.

that encompass the processing delays and queueing delays encountered by packets while they are passed between the host network device to the network device drivers of the guest. As can be seen in Figure 3.4, we consider the time taken by the packets to reach the virtual machine's network device drivers from the time they are received at the host network device and vice-versa, to constitute the latency due to virtualisation overheads.

The underlying mechanism of the VirtOCalc tool can be understood with the simplified algorithm in Algorithm 1. The tool carries out parallel packet captures on host and guest machines followed by transfer of the captured file from the guest and eventually the overhead calculations. The drift calculator tool is used initially by VirtOCalc to establish the clock offset between the host and the guest in order to facilitate accurate calculations. Once the packet capture is complete, they are compared packet by packet to calculate the latency contributed by virtualisation overheads to the overall network latency experienced by those packets.

Passwordless SSH mechanism must be established on the cloud setup beforehand, so that the tool can easily carry out packet capture remotely at the

Algorithm 1 Calculate latency due to VO for each captured packet

Input: $clockOffset, duration, guestInfo, captureFilter$
GuestPacketCapture($duration, guestInfo, captureFilter$)
HostPacketCapture($duration, captureFilter$)
for each hostPacket *in* host.pcap file **do**
    $guestPacket$ =GetCorrsPacketInGuestFile($hostPacket, guest.pcapfile$)
    $vo = hostPacket.time - guestPacket.time - clockOffset$
    StoreCalculatedVO($vo, hostPacket.protocol$)
end for
CalculateAverageVOPerProtocol()
Print *results*

guest machines. The Paramiko module of Python helps in coordinating the remote login using SSH while Scapy is used to capture packets and read the captured files. Object oriented paradigm has been adopted in the development of this tool, therefore, its classes can be easily enhanced and it can even be used as a module with other Python programs.

```
chauham1@ws-47:~/laghound/src$ python VirtOverheadCalculator.py 192.168.122.246 man
WARNING: No route found for IPv6 destination :: (no default route?)
WARNING: DNS RR prematured end (ofs=53, len=45)


Processed Packets:
        144 TCP Packets
        0 UDP Packets
        6 ICMP Packets

--------------------------------------------------------------------------------
                    Average Virtualization Overhead

    PROTOCOL         Avg. Latency Up          Avg. Latency Down
      TCP                0.19986                  0.11989
      UDP                  None                     None
      ICMP               0.13128                  0.07673
--------------------------------------------------------------------------------
chauham1@ws-47:~/laghound/src$
```

Figure 3.5: A snapshot of a sample output given by the VirtOCalc tool

The following are the key features of this tool:
Key Features

- Developed in Python, using Paramiko and Scapy modules. It can be run as a standalone program and can also be used as a Python module
- It is a command line based tool and it provides a formatted output as shown in Figure 3.5
- It is run on the cloud host machine and it coordinates packet capture at the host as well as the specified guest simultaneously
- It can be used to process packets captures online as well as offline by processing captured packets in .pcap file format
- Currently it can process ICMP, TCP and UDP traffic and give separate average latency values for each
- It can be configured to run for a particular time duration after which it gracefully exits after presenting the results to the user
- It is accurate to microseconds

3.4.4 Other Tools

The following tools and utilities were also used extensively during experiments.

- **Netperf** Netperf is a benchmarking tool that is widely used for measuring the performance of different kinds of networking. It enables measurement of metrics such as unidirectional throughput and also end to end latency for a connection. Furthermore, it supports measurements on protocols like TCP, UDP and SCTP for both IPV4 and IPV6 and also provides tests specific to them. We used Netperf version 2.6.0 for all our experiments.

  As netperf is suited for bulk data transfer measurements we used it extensively for VM to Host throughput measurements and vice versa. We used the TCP_STREAM and UDP_STREAM tests specifically for throughput measurements and used TCP_RR test for calculating the round trip times. Furthermore, it provides options to measure the CPU utilisation both at the sender as well as the receiver by setting the cC options. This helps in understanding the variation in CPU requirements for different buffer sizes and test durations.

- **Tcpdump** [2] is a free software distributed under BSD licence. It is a powerful, command line based packet analyser used for capturing and analysing packets over the network to which the computer is attached. It allows the user to sniff the network, that is, to intercept and print the packets being transmitted or received by the computer. It reads the packets from the network interface it is attached to or from a file containing previously captured packets. It can print the contents of the network packets and can print its output to the standard output or to a packet capture file. It also allows the users to configure a BPF based filters [18] to reduce the number of packets captured and displayed to the user in circumstances when there is a huge volume of network traffic.

  This tool was used for general packet capturing to analyse the network connections and also in specific cases in our experiments. The drift calculator tool and even VirtOCalc use Tcpdump for capturing packets on the remote virtual machines.

- **Ping** [3] is a networking utility that is used to test the reach-ability of remote computers on an Internet Protocol based network and also for measuring the Round Trip Time for packets sent to these computers from the local host. Ping is based on ICMP Echo request and reply messages. It transmits the echo request message to the remote computer and waits for the corresponding response. The time duration from the transmission to reception is measured as the Round Trip Time and any packet losses are also recorded. It gives the results in the form of a summary containing the minimum, average and maximum values of Round trip time measured along with the standard deviation for the same. It also shows the packet loss statistics if any. This tool is easy to use and can be employed to generate baseline RTT values for a particular connection quickly. It can also be used to quickly test whether the remote hosts are up or not.

- **Mpstat** [4] is a command line based tool for the Unix like operating systems that provides its users, statistics related to CPU usage on the screen. It is usually used to monitor the computer's performance and also to generate CPU related statistics for the same. We used this tool to record the CPU utilisation of each processor core of the Cloudlet Host while we conducted the networking performance tests.

[2] http://www.tcpdump.org/
[3] http://linux.die.net/man/8/ping
[4] $http://www.linuxcommand.org/man_pages/mpstat1.html$

We configured it to report CPU statistics per core at intervals of two seconds as the script coordinating the experiments ran simultaneously. Juxtaposing the results from the experiments with the CPU statistics generated using mpstat we can draw conclusions regarding the CPU requirements for different tasks.

3.5 Test Case Design and Experimental Scenarios

3.5.1 Classes of Test Cases

The tests were configured to provide optimal test coverage to elicit a satisfactory solution for all the research problems listed in Sec 1.1, Ch 1. The tests were designed to cover the following key areas:

- **Effect of Increasing Tenancy** In this test we analysed the network performance and bandwidth distribution by varying the number of tenant VMs on the host. We start with just the Test VM and then iteratively increase the number of VMs on the host whilst measuring network performance metrics in each case. This test case would also reveal how bandwidth is being apportioned amongst different VMs as the number of VMs increases. Furthermore, it would posit the variations in RTT as the tenancy increases at the Cloudlet. The aim was to keep increasing the number of active VMs till we had more VMs than the actual physical cores on the Cloudlet Host.

- **Effect of different CPU workloads** Here we ran a script to calculate *md5* hash of a large file on one of the VMs while carrying out a bulk TCP flow from the Host to measure the network throughput between Host and the VM. VirtOCalc was used to calculate the latency due to virtualisation overheads. The file size and the number of processes was varied to achieve different CPU utilisation rates. This class of experiments puts forth the relationship between CPU utilisation and network performance metrics on a KVM virtual machine.

- **Effect of Network intensive workloads** This class of test cases covers the tests involving parallel bulk data transfers to and from the Test VM and also the impact of hosting a Streaming server on the networking performance of the VM.

- **Effect of workloads on neighbour VMs** In this class of test cases we studied cross effects of workloads on neighbouring VMs on the networking performance of our Test VM. This is a rather broad class of experiments which is a superset of some other experiments from the list as well.

- **Effect of Cross Flows** This class of test cases dealt with the impact of network intensive workloads on neighbouring VMs such as video streaming and bulk data transfers, on the performance of the Test VM.

- **Effect of disproportionate VM allocation** In all other test cases the allocated VMs are symmetrical as far as their configuration is concerned. These test cases aim to study the networking performance of the Test VM when another VM with a much superior hardware configuration is allocated along with it on the Cloudlet.

- **Effect of Processor Pinning** In this case we pin the Test VM to a particular CPU core and then analyse its performance against the case when no CPU pinning was done.

- **Effect of Bandwidth Ceiling** In this case we analyse the bandwidth distribution amongst different VMs when a maximum throughput limit is imposed on the Test VM while rest of the VMs are free to throttle as much traffic as they can.

The following subsections discuss some of the test criteria and scenarios along with measurement techniques used for studying the impact of influencing factors on networking performance:

3.5.2 Bandwidth distribution at the Cloudlet

We used Netperf for analysing the bandwidth distribution patterns at the Cloudlet. LaCurts et. al[17] used TCP test of Netperf for a duration of 10 seconds between all VM pairs. This was done to determine the typical throughput for all inter VM paths in the network topology. Based on the typical throughput values we can determine if there is any rate limit being imposed on the VMs.

However, as all the VMs are hosted on the same host on the Cloudlet, we do not need to study the inter VM bandwidth metrics. Nevertheless, we need to focus on the Host to VM and the reverse path, as to how bandwidth is being distributed amongst different VMs and what are the factors that affect this

distribution. To this end, we used Netperf in our experiments and configured them to assess the bandwidth distribution as a function of traffic shape, CPU utilisation and VM configurations.

3.5.3 Cross traffic

This scenario is helpful in profiling the inter VM network as well as the Host to VM network. Idea is to estimate the number of concurrent bulk TCP connections on a path between two VMs and to analyse whether the number of other connections on the path remain stable or does it vary significantly with time. These tests aims to finding whether the path from

This can be achieved by sending Netperf traffic on both paths simultaneously. If throughput on the
However, this scenario is not relevant in our case as we are not dealing with a large scale cloud with multiple server racks where the VMs can be allocated on any of the servers. We have a Cloudlet hosted on a single physical server where all the VMs are effectively one hop away. Therefore, the bottlenecks would be either on the network interface of the Host or the Wifi connection from the mobile device to the host.

We wished to study the impact of the cross traffic from neighbouring VMs on the network bandwidth and the round trip times experienced by our Test VM. For this, we used netperf to generate bulk data transfers from the neighbouring VMs while we analysed the performance of the Test VM simultaneously.

3.5.4 Latency due to KVM Virtualization overheads

The following techniques were used to study this metric:

- **Packet Tracing** Our test setup comprises of several VM's bridged via a virtual bridge Virbr0 on the physical Host using KVM-Qemu. As the guest sends a packet the host kernel network driver injects this packet to the guest's tap device. The linux network stack then hands the packet from the tap device to the bridge which then forwards the packet to the device drivers of the guest VM. VHost-Net drivers read and place the packets into the virtio-receive buffer of the guest. Therefore, to analyse the time delays incurred due to this chain of packet transfers, we need to capture packets using Tcpdump at the capture points as illustrated in Figure 3.6. This idea is validated by Stephan Hajnoczi in one of his

replies to a question on capturing traffic between VMs, on the QEMU developer mailing list [13]:

- Source guest's tap device or network interface
- Bridge interface, virbr0
- Destination guest's tap device.

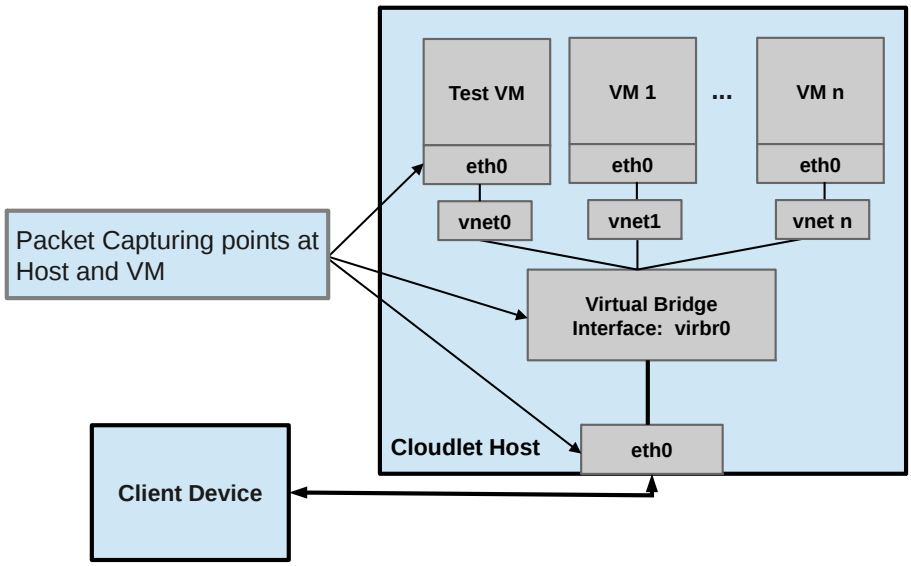

Figure 3.6: Packet Capturing points in the setup

Then comparing the packet capture files, we can calculate the time taken by the packets to reach the guest machine from the Host network interface through the network virtualisation middleware.

- **Using Ping or TCP Syn Packet arrival** To generate base line measurements for average round trip times and latency on te Host - VM network, we used Ping tool and also the TCP Syn packet arrival time.
- **Using function tracing** Function tracing tools like Ftrace[26] can be used to determine the time taken by the host kernel and network drivers of the guests to process the packets before handing them over to the receiver application. Function tracing shall result in an accurate

estimate of the latency component caused by the Virtualization overhead as it can reveal precisely how much time was spent by different intermediate Kernel functions in the whole chain of packet transfers. These tests are not necessary in the current scope of the work but may be carried out in future works.

3.5.5 Client Device to Cloudlet

This scenario is for the end to end networking performance measurement. In lieu of the mobile device, we used a workstation on the same LAN as the Cloudlet setup to act as a measurement end point or a client device for majority of our experiments. As we are solely focusing on the Cloudlet part of the network, we did not configure the following experiments that include the Mobile to Host network. However, these experiments can be carried out in future to provide the end user on the mobile device the information regarding the network state at the Cloudlet.

We need to install Tcpdump on a rooted mobile device to enable packet capture and offline analysis. Furthermore, we shall develop a Mobile application for determining the connection latency from the mobile device to the VM on the Cloudlet and to present to the user information about the WiFi latency and the latency caused by virtualization overhead at the Cloudlet. The server component shall run at the Cloudlet host that will analyse the packet traces and the connection itself to determine the time taken by the packets to reach the VM from the host and back and shall notify the mobile client of this overhead.

3.6 Test Suite and Sample Experiments

3.6.1 Test Suite

In all the test configurations our aim was to compute networking performance metrics such as, round trip time, throughput, one way latency and latency due to virtualisation overheads. We configured several scenarios on the Cloudlet adhering to the themes and classes of test cases, and carried out a battery of network performance measurement tests between the Host and the Test VM. Our test suite consists of the following tests:

- Netperf TCP_STREAM test with different send data sizes
- Netperf UDP_STREAM test with different send data sizes

- Netperf TCP_RR test to calculate successful Request Response pairs and RTT based on that
- Ping to actively get base line RTT
- VirtOCalc to get latency due to Virtualisation Overheads
- TRoTTEr to passively calculate round trip times of TCP traffic

Initially the test suite was run on the Test VM when all the neighbouring VMs were idle. This gave us base line values against which we compared the values obtained in the successive experiment. in every experiment the test suite was repeated 10 times and the resulting values of metrics were averaged to obtain one final result from each test of the suite.

Additionally we developed a ***Stress Test script*** that essentially carries out an $md5Hash$ of a random 1 MB file in an indefinite while loop. This allows us to emulate a CPU intensive workload that has a CPU utilisation rate of around 90-95%. Whenever we need to generate a CPU intensive workload on any VM, we run this script.

The following subsections describe some of the important experiments that were carried out which generated the results that have been presented in Chapter 4.

The following subsections discuss some example scenarios around which the experiments were designed:

3.6.2 Experiment: Hello Neighbours

This experiment belongs with the class of tests analysing the *Effect of Increasing Tenancy* on the Cloudlet. It was carried out on a Workstation that contained 4 CPU cores. Five Identical VMs were allocated on it, each configured with 1 vCPU and 1 GB memory. Initially, only the Test VM was active and the test suite was run on it. In the next iteration one more VM was made active and then the tests suite was run between Host and Test VM. This continued till the number of active VMs reached 5. Thus we kept on increasing the number of VMs by one in each iteration while continuously measuring the network performance metrics at the Test VM.

3.6.3 Experiment: Rate Limitation

In this experiment parallel bulk TCP flows were coordinated using Netperf from the Host to all the active VMs. The number of active VMs was 5 as

in the last experiment. For base line measurements we ran parallel Netperf flows without imposing any rate limitation on the Test VM. Once the results were obtained we limited the Test VM's network bandwidth to 200 Mbps. This was done by editing the bandwidth attribute of the network element in the XML file for the Test VM that was created by KVM-QEMU. Once again the same test was carried out and the results were noted to draw a contrast with the base line measurements.

3.6.4 Experiment: Different Workloads

This was the most elaborate experiment orchestrated by us and fortunately the most rewarding. We tried to emulate a real world scenario where the Cloudlet hosts servers running different types of workloads. We had 5 VMs in total, including our Test VM, 2 VMs running Video Streaming server using VLC player and 2 VMs running a $md5Hash$ command on a 1 MB random file infinitely. Figure 3.7 illustrates the scenario and VM allocation scheme on the Cloudlet for this experiment.

This experiment involved running the aforementioned test suite on the Host-Test VM link in the following different cases:

- Best Case: When all other VMs are Idle, to generate base line measurements
- General Case: When all the VMs are actively running their workloads
- Test VM and the Video Streaming Server pinned to the same CPU core
- Test VM and the Hashing Server pinned to the same CPU core

We wished to study the impact of cross flows and CPU sharing on the network performance of the Test VM through this experiment. This experiment was carried out on both our cloud setups separately and the results obtained correlated with each other.

3.6.5 Experiment: Asymmetric VMs

This experiment involved allocation of a micro instance VM as the Test VM along with a medium instance VM on the same Cloudlet Host. More specifically the Test VM has 1 core and 1 GB memory while the other VM had 4 cores and 2 GB memory. This experiment was carried out to understand whether a resource rich VM starved the micro instance by grabbing majority

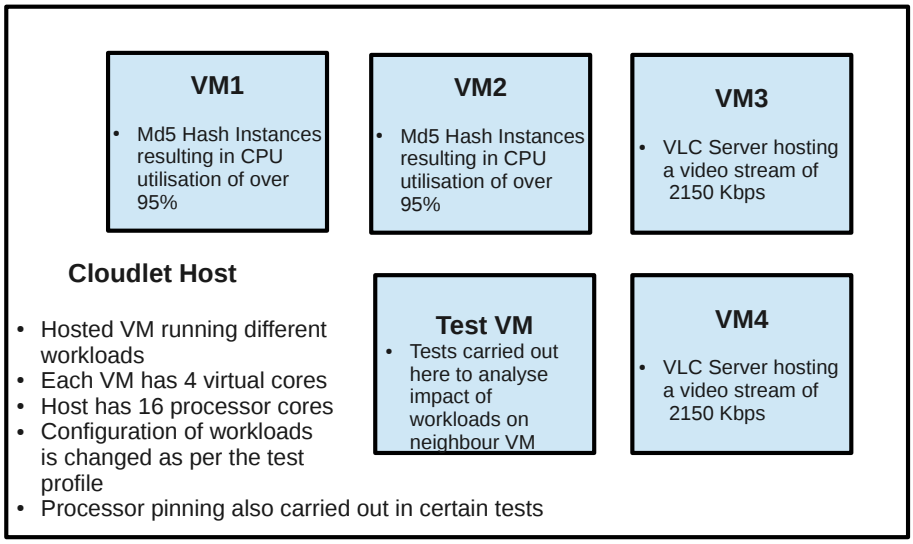

Figure 3.7: A sample VM workload profile on the Cloudlet host. In this configuration 2 VM are running streaming servers and 2 are running CPU intensive tasks. The test VM is just running the tests.

of networking resources. Test Suite was run as described before and the results were duly recorded.

3.6.6 Experiment: CPU load and Latency

In this experiment we allocated an instance with 4 vCPU for the Test VM and then configured it to run the Stress Test Script to increase the CPU utilisation on the VM. The Test VM has 4 vCPU and the hashing command in the script usually occupies one vCPU. Therefore, by increasing the instances of the hashing script we can increase the overall CPU utilisation of the Test VM.

Thus we carried out latency measurements on the Test VM at different levels of CPU Utilisation rates to draw a relationship between the latency and the CPU load on the VM.

3.6.7 Experiment: VO latency and different workloads

Four VMs of micro instance were allocated on the Cloudlet Host. Two VMs ran the $md5Hash$ command that led their CPU utilisation to 85-90%, while one VM hosted a Video Streaming server, and one Idle VM posed as the Test VM. VirtOCalc Tool was specifically run to calculate the bidirectional latency due to virtualisation overheads on all the VMs. This experiment was aimed to analyse the variation caused by different CPU loads to latency due to virtualisation overheads.

In this section we discussed how we carried out the experimentation, observation and analysis to put forth our findings and results. First step in this direction was setting up the test bed followed by the identification and definition of the metrics we wanted to study. Then, we proceeded to design different classes of test cases which would ensure a comprehensive test coverage for our experiments. Once the test cases were defined, we identified the measurement tools that could serve our purpose and even developed some turn key tools specific to our requirements. Finally, we configured the test cases to run on the available test setup, enabling us to collect data, observations and to formulate the concluding results. In the following section we discuss all the interesting results that we obtained from the rigorous experimentation and data analysis of this section.

Chapter 4

Results and Analysis

We carried out several rounds of experiments as described in the previous chapter and each class of experiments led to some interesting out takes. The following sections discuss the various experimental results that were obtained regarding network bandwidth distribution patterns at the cloudlet host, relationship between latency and CPU workloads and the contribution of virtualisation overheads to end to end network latency. Each section focuses on one particular aspect of the experimental results, and analyses it thoroughly to develop and provide a cohesive understanding of the findings.

4.1 CPU Utilisation and Bulk data transfer

Data transfer between clients and the server over the network depends on different factors like available network bandwidth, network round trip time, data buffer sizes at both ends and the processing power of the two Hosts. In a cloud computing scenario, guest VMs share the physical resources like memory, CPU, network and disk space of the Host server. Therefore, the performance of each virtual machine is dependent on the tasks running on it and also on the resource subscription by the neighbouring VMs.

In our experiments with bulk data transfer between the Host and the guest machines, we observed that bulk data transfer over TCP and UDP is a processor intensive task and it results in high CPU utilisation levels both at the Host and also at the VM. A key observation was regarding the affect of traffic shape on the CPU utilisation of the VMs. It was seen that with an increase in the size of individual application data packets, the CPU utilisation at the receiver VM drastically increased, see Fig. 4.1. On the other hand, both the Host and the VM's CPU utilisation was moderate when the application data

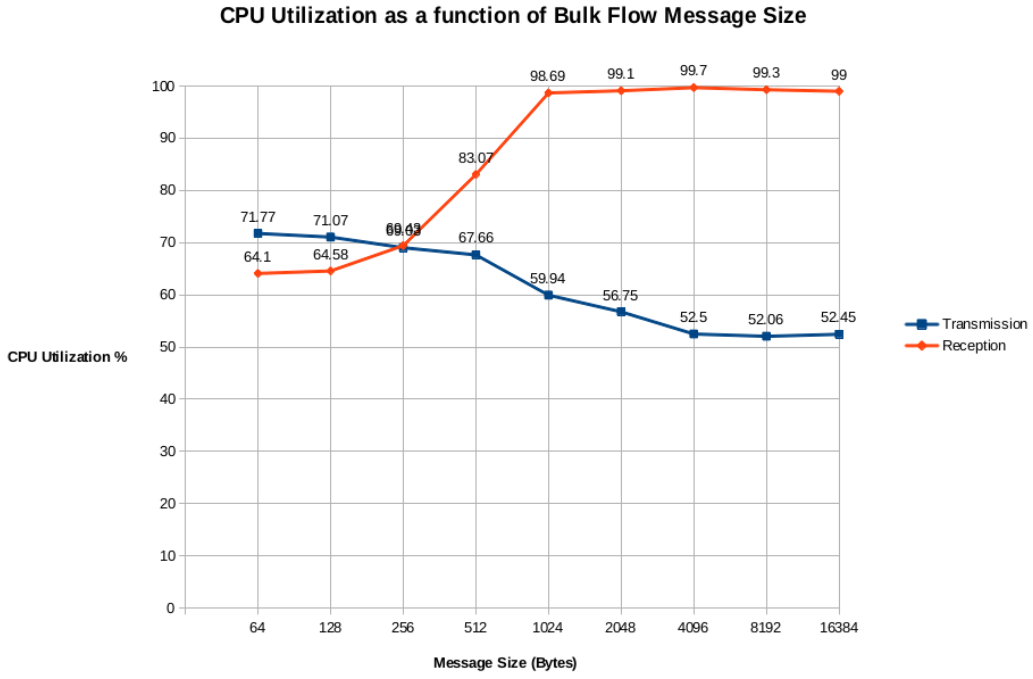

Figure 4.1: CPU utilisation at the sender as well as the receiver end for TCP bulk flows of different sizes. Netperf bulk data transfer was initiated from the cloudlet host machine to the test VM.

size was relatively smaller.

A possible explanation for this could be the added processing overhead for larger packets and even the need to reassemble fragmented packets at the receiver end. This is so as the default MTU for an IP packet over ethernet is 1500, which means lesser number of packets required to be processed for smaller message sizes. However, in case the application data size is larger, it would cause fragmentation of data into different IP packets which would then be reassembled at the receiver before passing it onto the application layer. Furthermore, as bulk transfer tests using netperf tool typically employ packet bursts at high speed, the combined tasks of correctly receiving packets over the network and simultaneously reassembling them to form full sized application data packets can definitely drive the CPU utilisation rate high.

Such traffic patterns are analogous with those of applications where a continuous stream of data is being transmitted between the clients and the application server. Therefore, a key finding from this experiment is that a VM such as an AWS EC-2 micro instance is definitely not suitable to host an application server that requires high networking performance along with decent compute performance. For such applications, VM instances with higher vCPU allocation must be commissioned so that the application's overall performance is not affected even if the networking requirements are high.

4.2 Host-VM Bandwidth distribution

The bulk data transfer experiments also gave a fair insight into the bandwidth distribution patterns for different VMs on the cloudlet host. We wished to analyse the effect of increasing the number of VMs on the bandwidth or throughput achieved by the test VM. For this, we calculated the maximum attainable throughput values for all the VMs active on the host simultaneously and after each iteration added one more virtual machine to the mix. For a setup having four processor cores, we went up to five virtual machines emulating AWS EC-2 micro instances. We ran the tests in two configurations, once with a rate limit and once without any specific ceiling on the maximum achievable bandwidth. We also wished to analyse the bandwidth distribution in relation to the traffic shape and application data size. The experiments gave some ineresting findings which have been discussed in the subsections that follow:

4.2.1 Case: Bandwidth ceiling not imposed on VMs

In this case we ran the tests without implementing any rate limitation on the test VM, therefore, all the VMs had an equal opportunity to burst through as much traffic as they could to achieve the maximum possible throughput. As we can see from Fig. 4.2, there is an upper limit to the maximum achievable bandwidth for the Host-VM link that is equally distributed amongst all the competing VMs. If there is only one VM active on the host, it may not always be able to grab the maximum achievable bandwidth. However, if more VMs are added, then they saturate the Host-VM link and make optimum use of the total available bandwidth. Furthermore, it was also observed that, just the act of adding more VMs or allocating resources to the VMs does not affect the performance of the Test VM until, the neighbouring VMs start using those resources for their tasks. As expected, the cloudlet host allocated

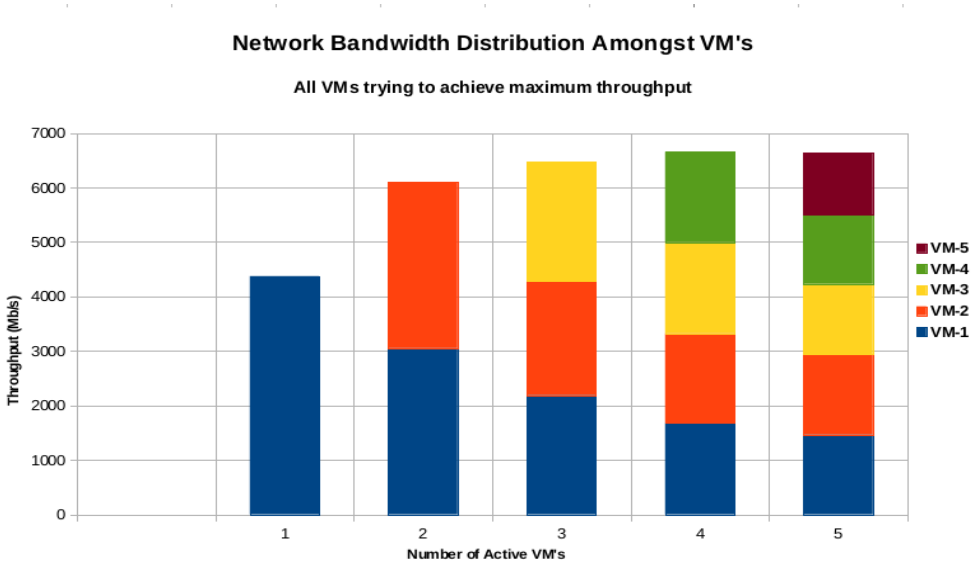

Figure 4.2: Throughput achieved by competing VMs when not rate limitation has been implemented. Each VM tries to achieve maximum possible throughput.

the bandwidth amongst all the hosted VMs fairly and equally.

It must be noted that for each additional bulk flow, the maximum achievable throughput for the test VM gets decreased proportionately. Let us consider Θ to be the bandwidth of the Host-VM link and θ to be the maximum throughput achieved by the Test VM. We can say that C is the number of cross flows currently active on the link where:

$$C = \lfloor (\Theta \div \theta) - 1 \rfloor$$

This observation follows the one made by Lacurts et. al. [17] for estimating the number of cross flows on a VM to VM path in a cloud. It is also quite useful in detecting a bottleneck link on the path between different VMs in an enterprise cloud deployment. However, it is not highly relevant in our current scenario of cloudlets where our VMs have been allocated on the same host machine. Nevertheless, the applications running on the VMs can make use of this information to make informed decisions regarding tuning their performance to ensure certain QOS guarantees to the clients. Examples of such informed decisions can be transmission of control messages to the client, regulating video quality and bit rate for a video streaming application.

4.2.2 Case: Bandwidth ceiling imposed on test VM

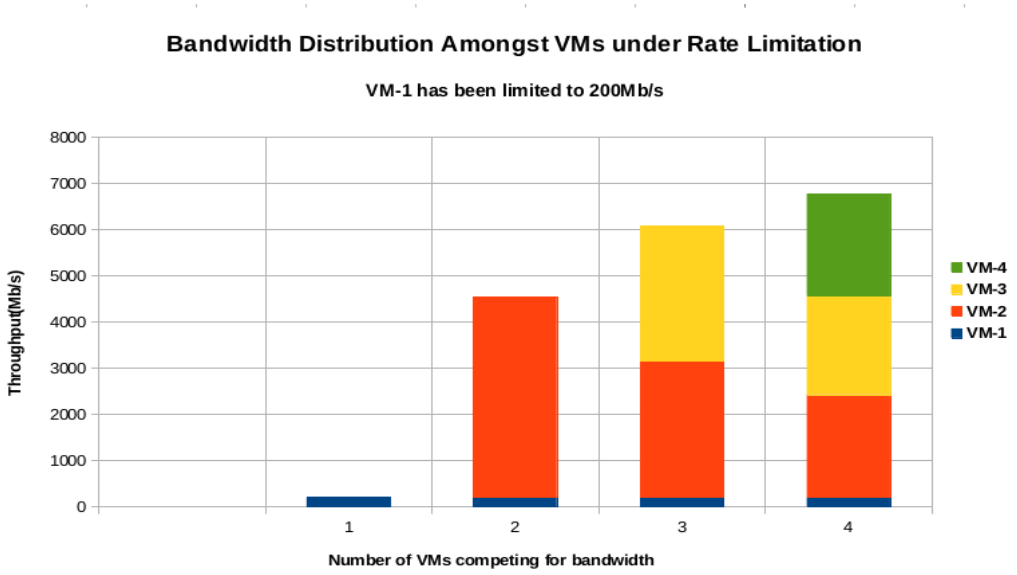

Figure 4.3: Throughput achieved by competing VM's when the test VM is rate limited to a maximum throughput of 200Mb/s.

In this case, we implemented a rate limitation on the bandwidth for the Test VM by editing the configuration xml file for the virtual machine. We stopped the running VM and then made changes to the xml file containing the settings and configurations that are used by KVM to initialise the VM. We changed the parameter corresponding to the network bandwidth to correspond to a rate limitation of 200 Mb/s.

In this configuration we had the rate limited Test VM and three other VMs which were free to burst through traffic at the maximum possible rate. It resulted in some interesting observations that can be seen in fig. 4.3.

Clearly, the Test VM did not exceed the stipulated limit of 200 Mb/s while the other VMs tried to achieve the maximum possible throughput. While the test VM stayed withing this bandwidth ceiling, the rest of the bandwidth was nearly equally distributed between the active VMs. This gives a clear indication that rate limitation is an effective mechanism of ensuring equity in bandwidth distribution, not just equality. The VMs that host applications which do not require a high amount of network bandwidth may be

rate limited so that other VMs that do need a higher bandwidth are free to achieve the same. This need based bandwidth allocation strategy may be particularly effective in a cloudlet scenario where all the VMs share the same Host-VM link.

A more effective strategy would be to put a blanket rate limitation as a policy on all the VMs. This way the cloudlet can avoid the perils of over subscription of network resources while being able to maintain a decent level of service quality. This strategy is implemented by IaaS service providers like AWS and Rackspace.

4.2.3 Case: Asymmetric, TCP Flows

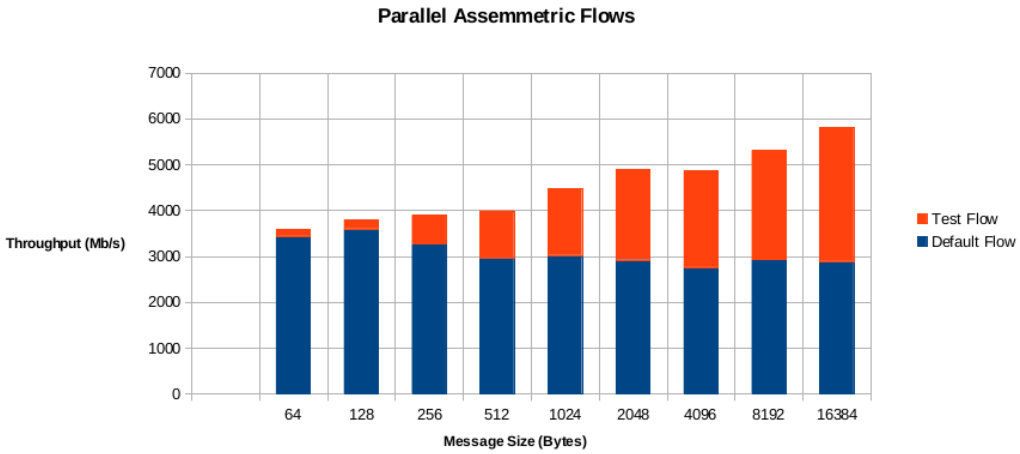

Figure 4.4: The bandwidth distribution amongst two competing asymmetric TCP bulk flows between Host and the VM. One of the flows is generated using the default TCP_STREAM test of netperf while the other flow's send data size was varied from 64 Bytes to 16385 Bytes

Till now we have considered bulk transfers where the data payload sizes were the same. In such cases, network bandwidth would ideally be shared equally amongst all the parallel flows, given the VMs are or similar configuration and are running equatable workloads. However, we wished to find out how does the data payload size affect the bandwidth distribution, keeping other factors same.

We observed that the Host-VM network bandwidth gets distributed amongst different TCP flows in direct proportion to the packet sizes being carried by them. In other words, as the payload size of the bulk TCP flows increases, they can capture more network bandwidth. This maybe partly be due to the fact that smaller flows do not saturate the bandwidth due to the limitations of TCP slow start. However, larger flows can easily saturate the TCP window size, hence achieving greater throughput.

These observations become increasingly relevant for a scenario where multiple VMs on a cloudlet host different services with contrasting data flows and traffic shapes. In light of this behaviour, it becomes necessary for the task placement framework to first profile the applications based on their possible traffic patterns and then place them in appropriately configured VMs.

4.3 Effect of CPU core sharing with neighbouring VMs

In order to study the effect of CPU core sharing between co tenant VMs, we carried out a series of tests and experiments on our testbed. As KVM used completely fair scheduling for scheduling the CPU access of guests, the guests with a higher workload will not be able to hog the CPU more than other guests. However, a scenario may occur wherein two guests have to share the same CPU core, and it may adversely affect the network performance of the guests depending on the type of workload.

To validate this supposition we ran different workloads on five micro instances of virtual machines on our Cloudlet and measured metrics like throughput and round trip times using netperf TCP_STREAM, UDP_STREAM and TCP_RR tests on the setup. We had an idle Test VM, two VMs hosting VLC media servers that ran 1450 Kbps video streams, and two VMs running file hashing scripts that saturated their CPU utilisation to 90-95%. The results from all these experiments have been presented through graphs with their explanation in the following sections.

4.3.1 Understanding Graph Legends

The experiments that we carried out for testing the effect of CPU pinning and different CPU and network workloads on the network performance of our Test VM resulted in rather complicated data. In order to simplify the

representation of the results we created graphs representing metrics against the Cloudlet configurations for that test. All graphs in this section bear the same legend.

In the legend, Base values represent the throughput values achieved when the host was in an idle state, i.e. the VMs were not running any CPU or network intensive workloads. Np processor pinning refers to the experimental results when the CPU scheduling task was left to KVM and no guest was pinned to any particular CPU core. In order to simulate a condition when the Test VM was forced to share CPU core with either the streaming server or the hashing server, we used KVM's processor pinning mechanism. Thus the two worms in the graphs, namely "CPU sharing with Hashing Server VM" and "CPU Sharing with Streaming server VM" represent the above mentioned configuration where the Test VM was pinned to a physical CPU core to which a streaming server or a hashing server was also attached to.

The results shown in these graphs reveal the network performance metrics for the Host to Test VM link which were being affected by the network and CPU loads of the neighbouring VMs.

4.3.2 Bandwidth distribution

Figures 4.5 and 4.6 illustrate different values of throughput achieved using different send data sizes from Host to Test VM and vice versa. It is to be noticed that this experiment's results too follow our previous observations that achieved throughput increases with the increase in the send data size and peaks at around the network MTU. Here, we can also see that the worm representing the base values registers the maximum throughput values amongst all the four configurations.

Interestingly, when all the VMs are active, running their respective workloads without any explicit processor pinning, the throughput results were better than the cases where the Test VM was forced to share CPU core with another guest. This validates the fact that KVM does indeed employ completely fair scheduling at it gives a good performance. We can also see that due to the CPU intensive nature of most of the workloads, the achievable throughout levels have considerably dropped for all cases when we compare them to the base values.

It must also be noted that in both the cases where processor pinning was used, the Test VM is unable to burst through its traffic completely and

achieves only a fraction of its peak performance. The case where it shares processor core with the hashing server displays the worst performance of all the configurations.

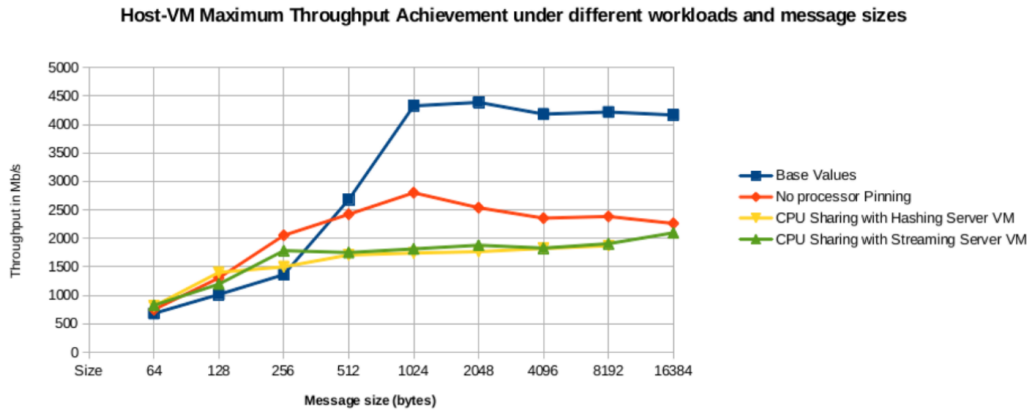

Figure 4.5: The maximum achievable throughput for a bulk TCP flow from the Host to the test VM under different workload scenarios and message sizes.

Similar observations can be drawn from the VM-Host path as well, however, the bandwidth achieved on this path is much lesser than those achieved on the opposite path. This can be attributed to the limited processing power of the Test VM when compared to the processing power of the Host. Here too, the case with CPU sharing with the Hashing server has the worst performance, with the case with no processor pinning representing the average case performance.

While the above mentioned experiments dealt exclusively with bulk TCP flows, we also carried out tests with UDP flows and the Figure 4.7 reveals the results from these tests.

It can be seen here that this graph also shows characteristics similar to those of the TCP test. It demonstrates steady upward growth curve showing and increase in achieved throughput with an increase in the send data size. It is worth mentioning that while we carried out the measurement of base line case, all the available CPU was being utilised for data transfer purpose exclusively with CPU utilisation rates upto 98% being observed at the Test

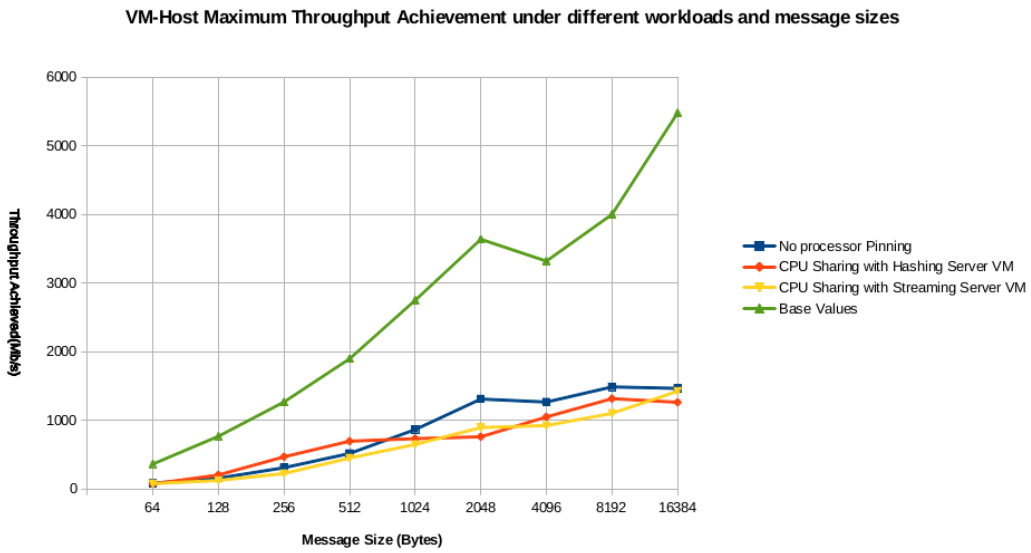

Figure 4.6: The maximum achievable throughput achieved for a bulk TCP flow from test VM to the Host under different workload scenarios and message sizes.

VM. However, this wasn't the case when we carried out experimentation with other configurations and CPU utilisation of 60-75% was observed for No Processor Pinning case and it dropped to as low as 48-60% for the processor sharing cases.

As UDP stream test of netperf is quite unreliable, there was a substantial difference between the sending rate and the actual reception rate throughput as reported by the test. Therefore, we have used the values for reception rates in the Figure 4.7.

4.4 Host-VM Request Response Test

Another interesting test that we carried out was the netperf TCP_RR test or the Request Response test. This test provides the number of successful request response transactions carried out in a fixed time duration during which the test runs. We ran this test for 10 seconds between the Host and Test VM under all the previously described configurations, turn by turn. Figures 4.8

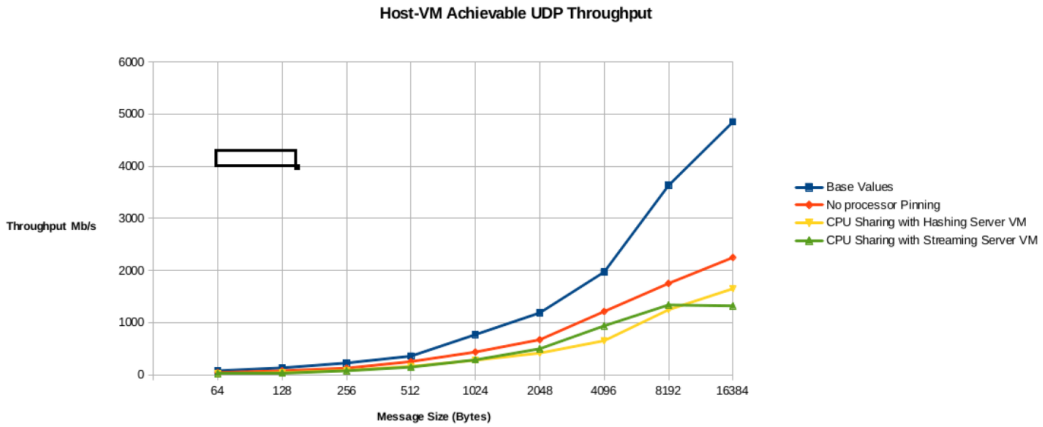

Figure 4.7: The maximum achievable throughput for a bulk UDP flow between Host and the Test VM under different workload scenarios and message sizes

and 4.9 illustrate the test results.

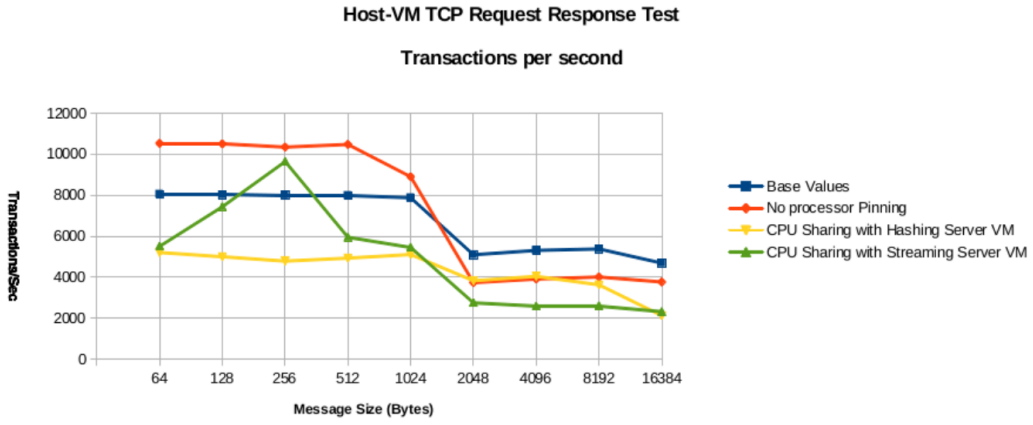

Figure 4.8: The number of request response transactions per second carried out between Host and the test VM

It must be remembered that higher the number of successful request response transactions per second, lower the value of network RTT on that link. Keep-

ing that in mind, we can see that the Host to Test VM path experienced substantial network delay when it shared CPU with either the streaming server or the hashing server. As the send data approached and elapsed the bitrate of the Video Stream being hosted by the streaming server, the performance of the VM started to degrade and it showed round trip times larger than any of the others.

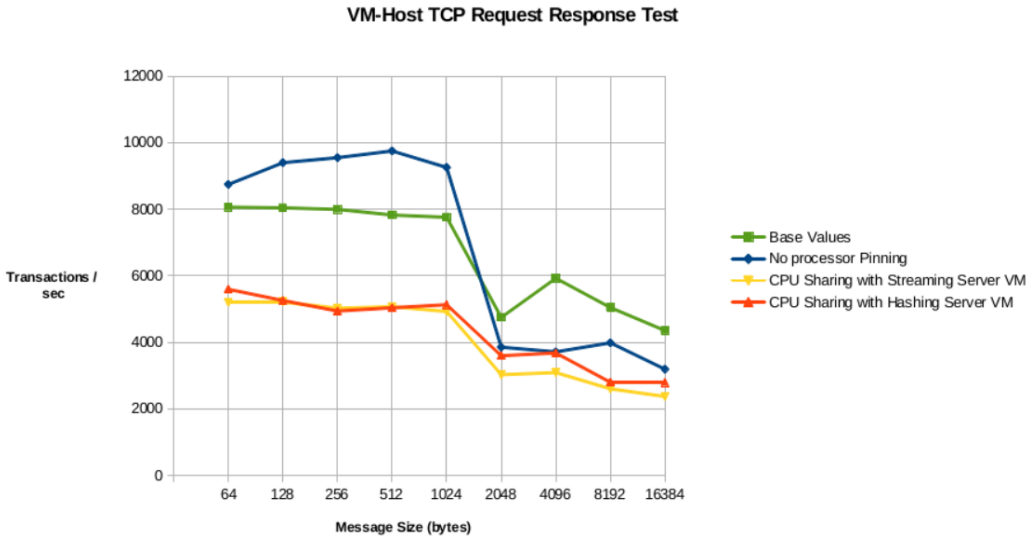

Figure 4.9: The number of request response transactions per second carried out between the Test VM and the Host

Similar observations were made on the reverse path as well, with the base values remaining relatively similar to the previous test. Surprisingly, the Test VM carried out more transactions while the other VMs were active, than when they were idle. However, this was true only for packets with send data size less than 1 Kb. As the send data size approached the MTU size or 1500 Bytes, the number of successful transactions fell, across all the test configurations.

The key result from these experiments was that the performance of the Test VM degraded due to processor sharing with other VMs with a heavier workload. The applications where latency must be low, should be hosted on Fixed performance instances which have dedicated CPU allocated to them. This would help reduce the performance hits that a VM has to incur due to

resource sharing and also due to the cross effects from neighbouring VMs.

4.5 Effect of CPU load on RTT

Thus far we have only studied and analysed the relationship between the guest CPU workloads and the throughput achieved by the guest virtual machines. In this section we will discuss the relationship between the CPU utilisation of the guest and the round trip times on a Host-guest connection.

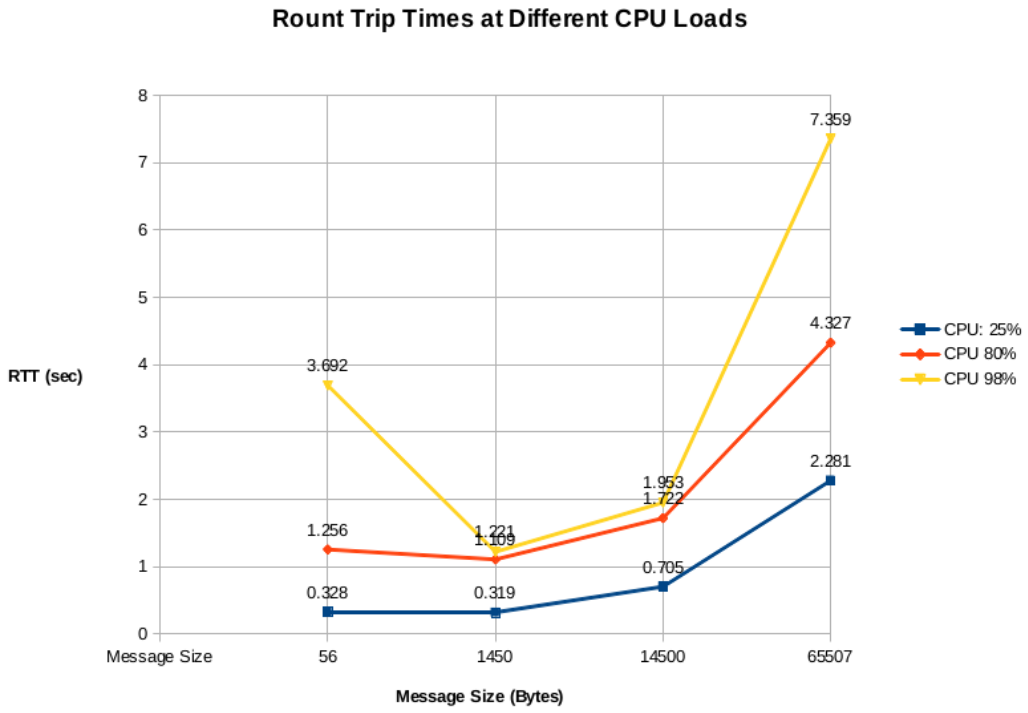

Figure 4.10: The variation in RTT between host and the Test VM under different CPU workloads on the Test VM

We configured workloads on the guests in such a manner that would give us a CPU utilisation of 25%, 80% and 98% on them. We can call these guests lowly loaded, moderately loaded and heavily loaded respectively. Therefore, we had three test configurations that gave us the results as expressed in Figure 4.10. We can see that the guest with the lower CPU consuming workload also has the least amount of round trip times for all send data sizes. On the

other hand, the heavily loaded guest showed RTT times as high as 300% of the RTT values obtained from the guest with a low workload.

It is worth noticing that the extreme values of RTT are observable for send data sizes of 56 Bytes and 65507 Bytes. However, for send data sizes closer to the network MTU (1500 Bytes), the observable RTT is quite similar for the moderately loaded and the heavily loaded guest. This means that although, RTT grows with the increase in the send data size, for best overall performance we must try to shape the traffic in such a manner that most of the packets contain a send data size closer to the network MTU.

4.6 Effect of Virtualisation overheads on network latency

As we have mentioned before, analysing the effect of virtualisation overheads on the networking performance and more specifically the end to end network latency is an important thrust area of our work. Through targeted test cases we aimed to understand to what degree is the end to end latency affected by virtualisation overheads. We employed packet capturing and specific tools like VirtOCalc to this effect. We found out that latency due to VO comes into play on both directions of the path. It is present on the link from the Host to the VM and also on the reverse path.

4.6.1 Contribution to the network RTT

As can be seen in Figure 4.11 the overall round trip time that is shown in green colour has a major contribution from the Average total VO latency, shown in yellow. This in turn is the sum total of the average latency due to VO from VM-Host and the average VO latency from Host-VM. This figure reveals that the latency due to virtualisation overheads may contribute upto 70-75% to the average round trip time experienced by the network packets on that path. We also measured the RTT values from the client machine on the LAN to the host server and these values were also of a similar order of magnitude as the Host-VM RTT values. As the client machine and the Host machine are on the same network, it is expected that the end to end latency for such a scenario would be quite low. In a Cloudlet scenario, we can also have a wireless connection, although, that also has to comply with the requirement of the Cloudlet server being within a couple of hops

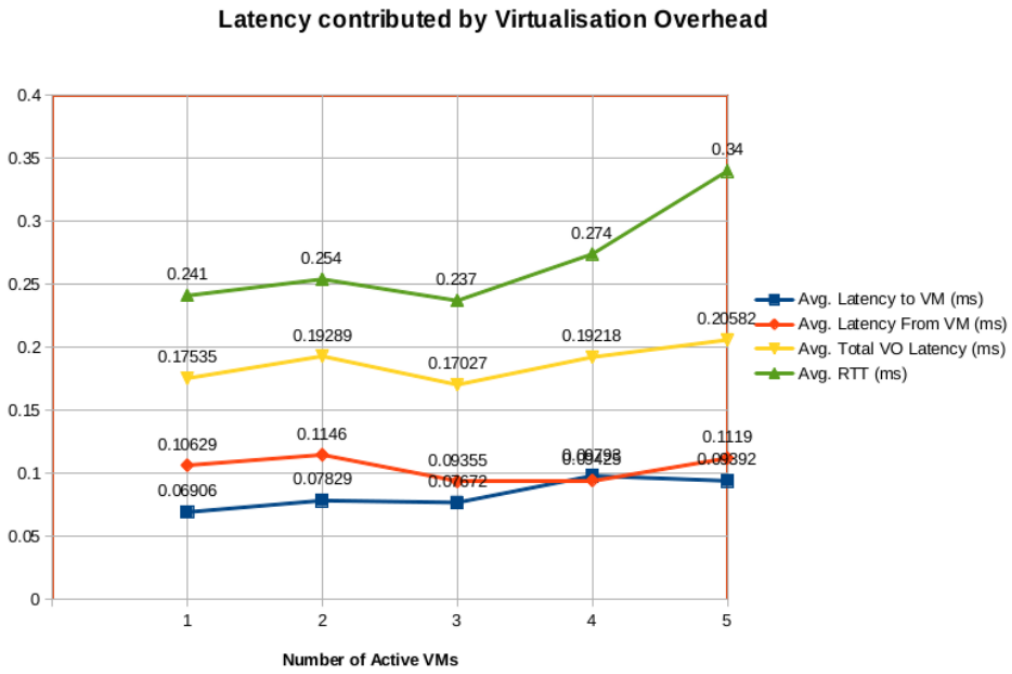

Figure 4.11: Host to Test VM Average round trip time and its constituent components of latency due to virtualisation overheads.

from the client devices. In both the cases, ideally the RTT from the client to the server is within a few milliseconds at max. However, it may increase due to an increase in the workload at the server or due to network congestion.

It must be noted that this is a base line figure, and has been generated from the experiment results at low workload profile on the guests. The latency due to VO may also increase significantly due to an increase in the workloads on the guests or even due to the over-subscription of resources by the neighbouring VMs. This is out next point of discussion.

4.6.2 VO Latency and CPU load

We have shown that virtualisation overheads play a significant role in the Host to VM latency, however, we wished to find out whether this latency was fixed or did it change with a change in workloads at the guests. Figure 4.12 tabulates the different values of latencies due to VO for different workloads running on the guests. For this we ran four guests, one hosting

VM Workload	Avg. Host to VM latency due to VO (ms)	Avg. VM to Host latency due to VO (ms)	Avg. Latency due to VO in the Round Trip (ms)
Streaming Server 1-5 % CPU	0.85241	0.07748	0.92889
85-90% CPU	1.11525	0.56984	1.68509
85-90% CPU	0.97238	0.48242	1.4548
Idle	0.85851	0.53902	1.39753

Figure 4.12: Latency due to virtualisation overheads for VMs running different workloads

a media streaming server, two hashing servers and one idle Test VM that only opened a TCP connection to the host for carrying out testing using the VirtOCalc tool.

As expected, we observed that the media streaming server showed low levels of CPU utilisation when compared with the hashing server that used upto 90% of the CPU available to them. The corresponding latency values for the above two cases also followed the same trend, as both the hashing servers exhibited an increase in the latency due to VO.

Even Idle VM may suffer from increased latency due to resource sharing with busy VMs. . This can be attributed to the cross effects of neighbouring VMs, because the other VMs ran CPU heavy tasks which might have affected the performance of the idle VM. As the Cloudlet Host was already heavily loaded due to the workloads on the hashing servers and also the streaming server, the addition of an extra VM meant that it must share the CPU core with either of the servers. This as we have seen earlier, degrades the performance of the guest, hence the increased latency due to VO on the idle VM as well.

We have seen in the previous subsection that the latency due to virtualisation overheads is the dominant component to latency between the host and the virtual machines, which eventually has some bearing on the end to end latency of the client to server connection. If the network is not congested, yet we experience increased levels of round trip times for the client to server connections, it can be attributed partly to latency due to virtualisation overheads along with the application server being heavily worked.

However, if the geographical distance increases (Hop count) the contribution of latency due to virtualisation overheads to the overall round trip time diminishes considerably. Only for a short distance communication, as is the model in Cloudlets, does latency doe to VO play any significant role, and that too is quite limited as the connection by design is quite fast anyways for most of the mobile cloud applications.

4.7 Cross effects of Heterogeneous tenants

VM Type	Base Line Throughout (Mb/s)	Avg. Virt. Latency Idle Case(ms)	Parallel Flow Throughout		Avg. Virt. Latency Busy Case (ms)
			Normal Case	CPU Heavy Task	
Medium Instance (4 Cores, 2 GB)	4272.85	0.21438	3184.38 (CPU 23 %)	3018.14 (55 % CPU)	2.00614
Micro Instance (1 Core, 512 MB)	4327.16	0.20138	3176.53 (CPU 90 %)	1603.28 (99.8 %)	0.91031

Figure 4.13: Bandwidth distribution and VO latency for a micro and a medium VM instance. This table shows how bandwidth allocation and latency due to VO is affected by running a similar workload on both VMs

While most of our experiments dealt with understanding the factors affecting networking performance of VMs that had identical configurations, we also experimented with scenarios where the contending VMs had different configurations. Figure 4.13 tabulates the results obtained from our experiments on two guests, emulating the micro and medium instances respectively. The maximum achievable bandwidth remained relatively similar for both the guests. Even when competing for network bandwidth with each other via netperf TCP stream tests, they managed to achieve almost equivalent throughput.

However, as we have seem in our previous experiments regarding the effect of CPU workloads on the bandwidth allocation, here, this observation became quite stark. We noted that when we ran a CPU intensive workload (hashing a large file) on both guests, the network performance of the micro instance dropped significantly. This was expected because its CPU utilisation rose upto 99.8% and as we have seen before, it results in degraded networking performance.

Furthermore, it was observed that bandwidth distribution remained fairly even till both the guests were moderately loaded. Once we ran the CPU intensive task on both, the limitation of a single vCPU of the micro instance was exposed and it could only achieve half of the throughout levels it had achieved earlier. It was also noted that the latency due to virtualisation overheads also significantly increased during this time.

Chapter 5

Discussion

Mobile Application developers usually implement certain application layer measures to combat the threat of occasional increased network latencies their services may suffer from due to uncertainties in the Cloud infrastructure. However, this assumes that an acceptable level of performance would be delivered by the cloud infrastructure too, otherwise, even the application level measures cannot guarantee a good service. Therefore, identifying performance bottlenecks and their timely removal is a critical activity for the cloud providers. We aimed to relieve them of this travail by trying to identify and understand the factors affecting networking performance of a cloud setup and their interrelationships.

5.1 Factors Affecting Cloudlet Networking Performance

5.1.1 CPU Utilisation Rates

All of our experiments verily brought to light the direct relationship between the CPU utilisation rate and the networking performance. We noted that round trip times between the client and the VM on the Cloudlet, suffered massively as the CPU utilisation rate at the VM increased to 90% from 25%. While such behaviour was expected, because CPU cycles are necessary for the execution of every instruction in a compute, we had not expected the incredibly high performance hit that was observed.

Furthermore, it was observed that VMs running CPU intensive workloads, failed to achieve the same throughput levels as they did under lighter loads.

This was also compounded by the apparent data loss during certain experiments while trying to burst through UDP datagram traffic. This direct correlation between CPU utilisation and dipping networking performance indicates that for latency critical applications, standard VMs like the AWS EC2 micro or Openstack Tiny instances are not sufficient. For better performance guarantees, the number of allocated vCPUs has to increase and the overall load must be moderate.

CPU utilisation rates can be used as a thumb rule for task placement as well. If the VM that we procured from any Cloud IaaS service shows high CPU rates even with trivial tasks, this might mean that the physical server hosting the VM is overloaded with workloads from other guests. We can request for a new VM hoping it would be allocated on a different server.
CPU utilisation rates also affect the equitable distribution of network bandwidth at the Cloudlet. Ideally there is fair distribution amongst all competing VMs, however if their is a contrast in their individual CPU utilisation rates then bandwidth gets distributed in an inverse ratio to it. Meaning, the lightly loaded VM will grab more bandwidth than a heavily loaded one.

5.1.2 Receiver Side Scaling

Most of the contemporary operating system's networking stacks provide support for multi-queues receives and Receiver Side Scaling [32]. This distributes network receive processing across several hardware-based receive queues, allowing inbound network traffic to be processed by multiple CPUs. RSS can help in reducing the network latency by reducing the overload on a single CPU while processing receive interrupts.

Therefore, for applications that put a high value on networking performance and latency and bandwidth guarantees, we must advocate the allocation of VMs with at least 4 vCPUs so that a single vCPU does not get overloaded, while the processing burden can be shared by others due to RSS mechanism. We can see how this ties up with the vCPU allocation strategies in the following paragraphs.

5.1.3 vCPU allocation strategies

Non Uniform Memory Access architecture of modern processors enables the individual CPUs to process their own local memory faster than non local memories either belonging to other CPUs or a shared pool. While this strategy provides enhanced performance for certain workloads, it can also cause

a bottleneck in cases when an application requires high memory bandwidth, but the data is stored on remote nodes.

This problem becomes evident in a virtualised scenario where VMs are allocated logical vCPUs that can be scheduled to run on any of the available physical CPU cores by the Kernel. This mechanism can explain why some VMs do not show high level of performance metrics even though they have been allocated multiple vCPUs. Let us say, we have a Test VM that has 4 vCPUs allocated to it. However, when its vCPUs get scheduled, they get scheduled to physical CPU cores that may not share the common memory pool or might not be on the same processor unit. This would result in a scenario where more time is taken than normal for processing receive interrupts or to carry out number crunching on large data sets. The overheads incurred in moving data to the local memories of the physical processing cores, every time the vCPU get scheduled on it would definitely cause a performance hit.

A solution for it is to pin the vCPUs to the physical CPU cores that share the same memory pools. This can be done easily in KVM by modifying the XML files specific to the allocated VMs. This way, the VM can exploit the faster processing of local data paradigm and attain high performance rates. Optimal use of processor pinning can also avoid CPU sharing between two VMs with contrasting workloads, which might result in reduced performance of the VM with the lighter workload due to cross effects from the other.

5.1.4 Multi tenancy

Although most of the virtualisation experts share the opinion that a CPU core can support upto 10 VMs successfully provided each VM runs at a CPU load of 10%. While this is theoretically possible, real life situations do not mirror the same. As the number of VMs grows on a physical host, each VM starts taking a performance hit due to the resource sharing and scheduling overheads. Even the networking resources like bandwidth is equally shared between the VMs provided the CPU workloads of the VMs are comparable. We have observed that if each VM has a considerable amount of workload, it is imprudent to allocate more VMs than the available CPU cores on the physical host. We can provide assured levels of quality of service by provisioning dedicated CPU cores to the VMs.

5.1.5 Cross Flows and Cross Effects

We term the performance hit taken by a VM due to the workload profile and configuration of the neighbouring VMs as cross effects. Cross flows or the concurrent data transfers being carried out by neighbouring VMs affects the round trip times as well as the throughput of the Test VM by saturating the VM-Host link and adding queueing delays at the same. We saw that cross effects may cause an increase in the latencies experienced by the Test VM and also eats into the available network bandwidth for the same.

5.1.6 Virtualisation Overheads

Virtualisation overheads are essentially contributed by the joint effects of the above mentioned factors. However, we have separately quantified the bidirectional latency due to virtualisation overheads as a performance metric in our work. It was observed that such delays play a significant role in the Host - VM data transactions, however, they may not have a major impact on the overall round trip time in commercial clouds. Nevertheless, our major focus is on Cloudlets, which are located one hop away from the client devices, and in such a topology, latencies due to VO may pplay a significant role. This latency reflects the workloads on the guests, as the guests running CPU intensive tasks most definitely display higher latencies. We even observed significant amount of latencies on some Idle VMs, which however, were attributed to cross effects from neighbouring VMs.

5.2 Takeaways for a Cloud Provider

Our results make for an interesting reading from a cloud providers point of view as it gives insights on how to fine tune VM allocation strategies and resource allocation mechanisms to attain maximum performance. Following are the major takeaways from our work that might be useful from a cloud provider's point of view:

- Bandwidth Limits must be imposed on VMs for an equity in performance. This would enable even the heavily loaded guests to achieve an acceptable level of throughput.
- For larger VMs, Processor Pinning should be used to allocate dedicated CPU cores to VMs for a guaranteed level of performance to certain customers. This would have two fold impact, one by supporting the receiver side scaling mechanism and secondly by preventing negative

effects of sharing vCPU with neighbouring VMs that run CPU intensive tasks. This can be packaged as a premium service or could be used to improve the overall performance of the service.

- For tiny instances or micro instances, no processor pinning should be used, and the scheduling must be left to KVM.
- Latency due to VO is a useful health indicator for the Cloud. It may be passively monitored continuously to set off alarms when it crosses certain thresholds. In such conditions, certain VM allocations may be rescinded or more hardware resources may be added.
- Workload profiling must be carried out for the applications before allocating appropriate VM configurations for them. Actually this is the job of a task placement framework, although, it could also be included by the Cloud service providers to improve the performance of the cloud.
- For customers craving high networking performances, their VMs must be allocated on dedicated hardware and as far as it is possible, VMs of similar configuration and belonging to the same task group must be allocated on the same physical host to avoid inter VM communication overheads/latencies and cross effects from higher configuration VMs.

Chapter 6

Conclusions

Mobile cloud computing carries the hopes and promises for ground breaking and cutting edge innovations in the sphere of mobile applications. Many exciting applications grounded in concepts of augmented reality and intelligent systems require a high performance network connectivity to meet the user's quality and usability requirements. Furthermore, ever increasing demands and expectations of users from mobile services, emphasise the need for measures to reduce end to end latency and service optimisation. Application level measures are the prerogative of the application developers, however, it is the primary responsibility of the Cloud Service providers to offer a level playing field in the context of networking performance.

We have been introduced to innovative concepts like Cloudlets, which aim to bring the computational resources to within a single hop from the client devices. Such measures result in reducing the physical distance that information has to travel, thereby, reducing end to end latency by a great degree. However, Cloudlets are also limited by the amount of physical resources available, because they are miniature cloud setups at best and can only serve a limited number of clients.

We focused on unravelling the performance and usability of KVM as an efficient full virtualisation solution for a Cloudlet Setup. It is endorsed by big wigs like IBM and Red Hat, already used by Openstack for Cloud provisioning and commercial services like Crosspeer Cloud are based on KVM. We were quite satisfied with the results as not only was VM provisioning and configuration quite straightforward, the design of KVM also ensured a good performance.
On the KVM based cloudlet we thoroughly analysed the various factors that have a direct or indirect impact on networking performance of a Cloudlet

and found interesting correlations between the CPU utilisation metric and the networking performance. We devised a test plan for extensively testing the setup with an aim of maximising test coverage. We designed and configured test cases to reflect real world scenarios as well as certain boundary cases for our own analysis.

We observed a direct dependence of achievable throughput and round trip times on the amount of CPU resources allocated to the VM. The busier the VM, the lower the performance metrics values became. The centrepiece of our analysis, Latency due to Virtualisation Overheads represents the indirect effects of overheads incurred due to virtualisation on the network latency. It proved to be an efficient health indicator for the Cloudlet as well as a measure of decreasing networking performance of the VMs.

While we observed a fairness in bandwidth distribution amongst similar VMs, there was also an indication that provisioning asymmetric VMs on the same physical host may deteriorate the performance of the tiny instances due to cross effects. We also discovered the impact of workloads running on neighbouring VMs on the performance of newly allocated VMs and suggested measures to prevent such situations. We also studied the impact of processor pinning and dedicated resource allocations on the overall networking performance of the VMs. We understood that in order to achieve optimum performance, we need to profile the applications running on the servers and then provision the best suitable configuration for the profiled workloads. Furthermore, we advised the Cloudlet providers to increase tenancy based on the performance metrics of the hosted VMs, so that issues arising out of resource over subscription may be avoided.

6.1 Implication of this work

Cloudlets are aimed at betterment of end to end networking and user perceived performance for certain novel innovations in mobile cloud computing. Usually the service level agreements of the cloud providers emphasise on uptime and available network bandwidth, but are silent with respect to network latencies and round trip times. We have attempted to correlate the relationships between various factors that may affect networking performance of a Cloud setup and this study reveals some ethical dilemmas.

We have shown that the VM's performance is affected by workloads on its neighbours as well as the Host machine's overall workload. This poses ques-

tions on the high level of virtualisation that is employed in clouds generally as all tenants may not be guaranteed similar level of services. The best way to ensure a good performance is to provision dedicated instances for customers, but this would mean a diminished return for the Cloud provider. It is imperative that the cloud provider sanction a level playing field for all users, however, they usually charge more for dedicated instances or cluster networking or instances that use the latest networking drivers to ensure high performance.

Armed with our results and general takeaways, the cloud providers can ensure certain levels of end to end latency and can also guarantee these levels in their service level agreements. An important application of this work is in optimising latency oriented VM provisioning for specific applications. The cloud providers can monitor their data centres and allocate VMs for low latency services on servers with less tenants and minimal overall workloads. This would ensure abstraction in the true sense with the customers not having to worry too much about procuring the right VMs for their services, as the cloud provider will automatically profile their requirements and provision a suitable VM on a suitable physical hardware.
An important aspect of this work is that it is derived from a research theme of Green ICT or energy efficient computing. Green computing ties with the concept of maximising the resource utilisation and carrying capacity of the physical devices. Cloudlets also provide efficient network connectivity for client devices while maximising the physical host's resource utilisation. Our work endeavours to find the causes of deteriorated network performance at the Cloudlet, which is necessary for maintaining a balance between maximising carrying capacity and guaranteeing certain levels of Quality of Service. Naturally this contributes towards ecologically sustainable development with respect to Green computing. Furthermore, with applications in optimising latency oriented task placement and VM allocation this work aims to enhance the efficiency and performance of the cloud setup and the mobile cloud applications utilising the infrastructure.

6.2 Future work

We focused solely on the Host to VM network behaviour assuming the transmission delays to be constant. This was to avoid running into the complexities of wireless transmission or network externalities that have already been studied extensively in literature. However, end to end networking perfor-

mance is dependent partly on the client to Host link and partly on the Host to VM link. Therefore, this work can be extended by reaching the performance metrics calculated at the Host to the Client devices, maybe through a dedicated mobile application. This would present the clients with the network health information of the Cloudlet as well as the VM it is served by.

We have also proposed to undertake analysis based on function tracing to quantify the virtualisation overheads in the form of latencies incurred in individual functions as network packets travel through the virtualised network stacks. Ftrace is a tool that may be used for such analysis.

Finally, the tools that we have developed for our research may be enhanced by adding new features or making them more robust. They may be combined to create a test suite that can effectively monitor the network health of the Cloudlet setup.

Bibliography

[1] Cfs scheduler. `https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt`. Accessed: 2014-02-24.

[2] John carmack's delivers some home truths on latency. `http://oculusrift-blog.com/john-carmacks-message-of-latency/682/`. Accessed: 2014-05-21.

[3] Akamai's state of the internet. Tech. rep., Akamai Technologies, Q1 Report, Volume 7, Number 1, 2014.

[4] Apparao, P., Makineni, S., and Newell, D. Characterization of network processing overheads in xen. In *Proceedings of the 2Nd International Workshop on Virtualization Technology in Distributed Computing* (Washington, DC, USA, 2006), VTDC '06, IEEE Computer Society, pp. 2–.

[5] Arsene, A., Lopez-Pacheco, D., and Urvoy-Keller, G. Understanding the network level performance of virtualization solutions. In *Cloud Networking (CLOUDNET), 2012 IEEE 1st International Conference on* (Nov 2012), pp. 1–5.

[6] Bahl, P., Han, R. Y., Li, L. E., and Satyanarayanan, M. Advancing the state of mobile cloud computing. In *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services* (New York, NY, USA, 2012), MCS '12, ACM, pp. 21–28.

[7] Bahl, P., Philipose, M., and Zhong, L. Vision: Cloud-powered sight for all: Showing the cloud what you see. In *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services* (New York, NY, USA, 2012), MCS '12, ACM, pp. 53–60.

[8] Campbell, A., and Choudhury, T. From smart to cognitive phones. *IEEE Pervasive Computing 11*, 3 (July 2012), 7–11.

[9] Che, J., Yao, W., Ren, S., and Wang, H. Performance analyzing and predicting of network i/o in xen system. In *Dependable, Autonomic and Secure Computing (DASC), 2013 IEEE 11th International Conference on* (Dec 2013), pp. 637–641.

[10] Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., and Bahl, P. Maui: Making smartphones last longer with code offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services* (New York, NY, USA, 2010), MobiSys '10, ACM, pp. 49–62.

[11] Grigorik, I. Latency: The new web performance bottleneck. `https://www.igvita.com/2012/07/19/latency-the-new-web-performance-bottleneck/`, 2012.

[12] Guan, H., Dong, Y., Ma, R., Xu, D., Zhang, Y., and Li, J. Performance enhancement for network i/o virtualization with efficient interrupt coalescing and virtual receive-side scaling. *Parallel and Distributed Systems, IEEE Transactions on 24*, 6 (June 2013), 1118–1128.

[13] Hajnoczi, S. Sniffing traffic between vms. `https://lists.gnu.org/archive/html/qemu-devel/2013-10/msg01254.html`, 2013. Source: Qemu-Devel mailing list.

[14] Jacobson, V., Braden, R., and Borman, D. Tcp extensions for high performance, 1992. RFC 1323.

[15] Kivity, A., Kamay, Y., Laor, D., and Lublin, U. amd Liguori, A. Kvm: The linux virtual machine monitor. In *Proceedings of the Ottawa Linux Symposium* (OLS '07, Ottawa, Ontario, Canada, July 2007), pp. 225–230.

[16] Kosta, S., Aucinas, A., Hui, P., Mortier, R., and Zhang, X. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proceedings IEEE* (March 2012), pp. 945–953.

[17] LaCurts, K., Deng, S., Goyal, A., and Balakrishnan, H. Choreo: Network-aware task placement for cloud applications. In *Proceedings of the 2013 Conference on Internet Measurement Conference* (New York, NY, USA, 2013), IMC '13, ACM, pp. 191–204.

[18] McCanne, S., and Jacobson, V. The bsd packet filter: A new architecture for user-level packet capture. In *Proceedings of the USENIX*

Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings (1993), USENIX Association, pp. 2–2.

[19] Mei, Y., Liu, L., Pu, X., Sivathanu, S., and Dong, X. Performance analysis of network i/o workloads in virtualized data centers. *IEEE Trans. Serv. Comput. 6*, 1 (Jan. 2013), 48–63.

[20] Menon, A., Santos, J. R., Turner, Y., Janakiraman, G. J., and Zwaenepoel, W. Diagnosing performance overheads in the xen virtual machine environment. In *Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments* (New York, NY, USA, 2005), VEE '05, ACM, pp. 13–23.

[21] Minnear, R. Latency: The achilles heel of cloud computing. `https://cloudcomputing.sys-con.com/node/1745523`, 2011. Accessed: 2014-02-05.

[22] Muraleedharan, R. Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In *Proceedings of the 2012 IEEE Symposium on Computers and Communications (ISCC)* (Washington, DC, USA, 2012), ISCC '12, IEEE Computer Society, pp. 59–66.

[23] Pelsser, C., Cittadini, L., Vissicchio, S., and Bush, R. From paris to tokyo: On the suitability of ping to measure latency. In *Proceedings of the 2013 Conference on Internet Measurement Conference* (New York, NY, USA, 2013), IMC '13, ACM, pp. 427–432.

[24] Popa, L., Krishnamurthy, A., Ratnasamy, S., and Stoica, I. Faircloud: Sharing the network in cloud computing. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks* (New York, NY, USA, 2011), HotNets-X, ACM, pp. 22:1–22:6.

[25] Ravindranath, L., Padhye, J., Mahajan, R., and Balakrishnan, H. Timecard: Controlling user-perceived delays in server-based mobile applications. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2013), SOSP '13, ACM, pp. 85–100.

[26] Rostedt, S. ftrace - function tracer. `https://www.kernel.org/doc/Documentation/trace/ftrace.txt`, 2008. Written for: 2.6.28-rc2, Updated for: 3.10.

[27] Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing 8*, 4 (Oct. 2009), 14–23.

[28] Song, X., Shi, J., Chen, H., and Zang, B. Schedule processes, not vcpus. In *Proceedings of the 4th Asia-Pacific Workshop on Systems* (New York, NY, USA, 2013), APSys '13, ACM, pp. 1:1–1:7.

[29] Soriga, S., and Barbulescu, M. A comparison of the performance and scalability of xen and kvm hypervisors. In *Networking in Education and Research, 2013 RoEduNet International Conference 12th Edition* (Sept 2013), pp. 1–6.

[30] Strowes, Stephen, D. Passively measuring tcp round-trip times. *ACM Queue 11*, 8 (Aug. 2013).

[31] Tao, D., Qin-fen, H., Bing, Z., Tie-gang, Z., and Li-ting, H. Scheduling policy optimization in kernel-based virtual machine. In *Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on* (Dec 2010), pp. 1–4.

[32] Tom Herbert (therbert@google.com), W. d. B. w. Scaling in the linux networking stack. Tech. rep., www.kernel.org.

[33] Van der Merwe, J., Ramakrishnan, K., Fairchild, M., Flavel, A., Houle, J., Lagar-Cavilla, H., and Mulligan, J. Towards a ubiquitous cloud computing infrastructure. In *Local and Metropolitan Area Networks (LANMAN), 2010 17th IEEE Workshop on* (May 2010), pp. 1–6.

[34] Wang, G., and Ng, T. S. E. The impact of virtualization on network performance of amazon ec2 data center. In *Proceedings of the 29th Conference on Information Communications* (Piscataway, NJ, USA, 2010), INFOCOM'10, IEEE Press, pp. 1163–1171.

[35] Whiteaker, J., Schneider, F., and Teixeira, R. Explaining packet delays under virtualization. *SIGCOMM Comput. Commun. Rev. 41*, 1 (Jan. 2011), 38–44.

[36] Zhuang, H. Performance Evaluation of Virtualization in Cloud Data Center. Master's thesis, Aalto University, School of Science, Finland, 2012.